Quantitative Robustness for Signal Temporal Logic with Time-Freeze Quantifiers

Bassem Ghorbel, and Vinayak S. Prabhu

***Abstract*—Signal Temporal Logic (STL) is a variant of Metric Temporal Logic (MTL) which can express intricate temporal requirements over signals, and has found wide adoption for expressing requirements over complex control systems models. A key factor in the success of STL has been that of quantitative robustness, and the development of efficient algorithms for computing the robustness values over traces. The real-valued quantitative robustness of a signal with respect to an STL property quantifies the degree of satisfaction or violation of the property by the signal. In this work we introduce a notion of robustness for a more expressive logic, Timed Signal Temporal Logic (TSTL), which can be seen as Timed Propositional Temporal Logic (TPTL) with predicates defined over real-valued signals. This logic can express many natural engineering requirements that STL cannot. We also develop algorithms for computing this robustness value over traces in the pointwise semantics. While the robustness computation for general TSTL formulae is PSPACE-hard due to the PSPACE-hardness of the monitoring problem for TPTL, for the special case of one variable TSTL, a fragment of TSTL which is still more expressive than STL, we develop an optimized algorithm which computes robustness in time *linear* in the length of the trace. Finally, we experimentally validate the tractability of our algorithms with our prototype tool in Matlab.**

I. Introduction

Temporal logics [5] are a rich formalism for rigorous specification of properties used for describing and reasoning about phenomena evolving over time; and are increasingly being used in the context of Cyber-Physical Systems (CPS) and Control [28], [39], [35]. Linear time temporal logics such as LTL allow talking about sequences of events in a given single execution with the help of temporal operators such as "eventually", denoted $\Diamond$, and "always", denoted $\Box$. A typical example of a property is $\Box\,(\mathit{req} \rightarrow \Diamond\, \mathit{grant})$; this property states that *always*, every request must *eventually* be followed by a grant. In CPS where timing requirements cannot be abstracted, temporal logics have been augmented with constructs for timing constraints, giving rise to *timed* temporal logics such as Metric Temporal Logic (MTL), Metric Interval Temporal Logic (MITL), and TPTL. MTL and related logics augment existing temporal modalities with timing constraints, for example, our earlier formula could be enhanced with timing constraints in MTL as: $\Box\,(\mathit{req} \rightarrow \Diamond_{\leq 5}\, \mathit{grant})$; which says that every request must be eventually followed *within* 5 *time units* by a grant.

The algorithmic development of temporal logics was originally motivated by applications in Computer Science under a Boolean semantics: either a system satisfies a property, or it does not. In the context of Cyber-Physical Systems, such a Boolean view is often restrictive – the mathematical models of CPS which are used to reason about behaviors are only an approximation of the actual systems, and hence asking for *exact* adherence to a logical specification in the Boolean sense is not in line with engineering practice where designers require quantifying *how well* a system satisfies given specifications. For example, quantitative analysis is a core component of linear systems design [29]. Much work has been done in recent years to lift formalisms which originated in the Boolean worldview to a quantitative setting [22], [30], [23], [20], [17], [8], [9], [7], [38], [4].

The development of quantitative semantics has opened up new avenues. In the case of temporal logics, the introduction of quantitative $\mathbb{R}$-valued semantics of MTL [17] led to black-box optimization based falsification testing of Signal Temporal Logic (STL, a variant of MTL over signals) specifications for complex industrial systems [17], [14], [13], [15], [37], [27], [43], [26], with tools such as STaLiRo, Breach, FALSTAR, and FalCAuN [18], [10], [16], [40]. A quantitative interpretation of STL formulae gives us a *robustness* function ρ, which for a formula φ, and a trace π, gives us a quantitative measure $\rho(\varphi, \pi)$ of how well the trace π satisfies/violates the specification φ. A negative robustness value implies violation of φ over π under the Boolean semantics, and a positive value implies satisfaction. Given this ranking function, one can employ black-box optimizers to search for a signal input such that the corresponding system output will have a negative robustness value for the STL specification. Efficient linear time algorithms have been developed for computing the STL robustness function over traces [13], [31], [10]. A strong point of this approach is that it scales to complex industrial models as models are viewed as black-boxes; as long as the model can be simulated efficiently, and STL robustness can be computed quickly, the framework can be used.

However, there are commonly occurring temporal properties that cannot be expressed in MTL/STL. For example, consider a property which says that whenever we have an e_1, then we must have a subsequent e_2, and then e_3, such that the time gap between e_1 and e_3 is ≤ 0.8 time units. MTL/STL either cannot express such requirements, or the requirement encoding is complex and non-intuitive. The issue arises when we have timing constraints with multiple temporal modalities in between. The logic Timed Propositional Temporal Logic (TPTL) [3] presented *freeze quantifiers* to express richer properties than MTL. Freeze quantifiers specifications subsume, and are strictly more expressive than, specifications in MTL containing only future temporal modalities; for example the MTL formula $\Box\,(\mathit{req} \rightarrow \Diamond_{\leq 5}\, \mathit{grant})$ can be written in TPTL as $\Box\,(\mathit{req} \rightarrow x.\Diamond\,(\mathit{grant} \wedge x \leq 5))$. The freeze quantifier cements the values of freeze variable to specific time values in the given execution so that it can be used later in time comparison predicates. While TPTL can express all of MTL, there

Department of Computer Science, Colorado State University (USA); email: bassem@colostate.edu, vinayak.prabhu@colostate.edu.
This material is based upon work supported by the National Science Foundation under Grant No. 2240126.

are formulae in TPTL that MTL, with only future modalities, cannot express [6], [24] (the situation is more nuanced when both future and past modalities are allowed [25], [34]). For the property we presented earlier, the corresponding TPTL formula can be written simply with just one freeze variable as $\Box x.\left(\left(e_1 \rightarrow \Diamond\left(e_2 \rightarrow \Diamond\left(e_3 \wedge x \leq 0.8\right)\right)\right)\right)$. The proof in [6] which demonstrates that TPTL is more expressive than MTL actually presents a one-variable TPTL formula that MTL cannot express; thus showing that one-variable TPTL is more expressive than MTL. Similar results hold for the logic variants over $\mathbb{R}$-valued signals rather than over atomic propositions, *i.e.* Timed Signal Temporal Logic (TSTL) and its one-variable fragment.

One can also take an automata theoretic approach for specifying requirements, typically this incurs an algorithmic cost as automata can be more expressive [11]. In the timed setting, timed automata are known to be strictly more expressive than MTL [33], [2]. Recent work [21] presented a monitoring algorithm for one clock timed automata that runs in time linear in the trace length, but with a multiplicative factor of $2^{O(|A|)}$ where $|A|$ denotes the timed automaton size. This large multiplicative factor makes a tractable implementation prohibitive. Alternative monitoring procedures based on the zone construction have been proposed for timed automata in [42], [41]. While these procedures have exponential time bounds, for the 1-clock examples in [42], the monitoring procedures were efficient. Our work considers only space robustness and not time robustness [14], [1], however time robustness notions can be developed for TSTL in our framework.

Our Contributions. This work is in the pointwise semantics where a timed execution is specified as a timed word: a sequence of timed observations of the system. We present four main contributions.

(I) We introduce and develop the notion of quantitative real-valued robustness for Timed Signal Temporal Logic (TSTL), the variant of TPTL over $\mathbb{R}$-valued signals. This logic is strictly more expressive than STL. Our robustness notion coincides with that of the commonly used robustness function for STL over the common fragment – given any STL formula φ, there exists an equivalent TSTL formula φ_{TSTL} having freeze quantifiers; for any trace π our robustness function on the TSTL formula φ_{TSTL} gives the same value as the commonly used robustness function from [31] on the STL formula φ. Additionally, matching corresponding results from STL, if the robustness value of a TSTL formula is positive, then the trace satisfies the formula; and if the robustness value is negative, the trace does not satisfy the TSTL formula.

(II) We present an offline algorithm for computing the robustness values of TSTL formulae over traces. This algorithm runs in time $O\left(|\pi|^{|V|+1}|\varphi|\right)$, where $|\pi|$ is the trace length, $|\varphi|$ is the formula size, and $|V|$ is the number of freeze variables. The exponential dependence on $|V|$ is not surprising given the PSPACE hardness of the monitoring problem for TPTL [32].

(III) For the special case of 1-variable TSTL, which is already more expressive than STL, we construct a novel algorithm which relies on a careful mixing of divide and conquer, and dynamic programming techniques to achieve robustness computation in *linear time*, similar to the algorithm in [19]. The linear time bound holds provided the trace has at most a constant number of sample points in any unit time interval, *e.g.*, for a trace obtained by a uniform time-sampling scheme. Given a timed word π, and a one variable TSTL formula φ, the complexity of our algorithm is $O\left(|\varphi| \cdot |\pi| \cdot \beta_{\max}\right)$, where $|\varphi|$ denotes the number of subformulae in φ, and $|\pi|$ denotes the number of timestamps in the timed word π, and $\beta_{\max}$ is the maximum number of timestamps in a window of time duration $r_{\max}$, where $r_{\max}$ denotes the maximum time constant in the formula (in case π has the integer timestamps $0, 1, 2, 3, \ldots$, we can take $\beta_{\max}$ to be $r_{\max}$). Note that for long traces which have a bounded skew, ensuring that only a constant number of sample points can arise in any unit interval, we have $\beta_{\max} << |\pi|$, and $\beta_{\max}$ to be independent of $|\pi|$, thus giving us a linear time bound on the algorithm. Our algorithm borrows ideas from the quadratic time monitoring algorithm of [12], our further developments and improvements lead to a linear time complexity (in case of bounded skew).

(IV) We implemented our algorithms directly in Matlab and not in C (on average, C is 50x faster than Matlab), and we present experimental results. While the general robustness computation procedure struggles over long words due to the PSPACE hardness of the problem, we show that our optimized algorithm for the one variable fragment of TSTL scales easily to timed signal words having tens of thousands of samples.

II. Timed Signal Temporal Logic (TSTL)

TSTL is a version of TPTL where instead of atomic propositions, we have signal variable predicates of the form $s \leq 5$ for a signal variable s. In this section we present the definitions pertaining to TSTL.

Signals,Traces. A $\mathbb{R}^n$ valued *signal* or a *trace* is a pair (σ, τ), where $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_{|\pi|-1}$ is a finite sequence of elements from $\mathbb{R}^n$, and $\tau = \tau_0, \tau_1, \ldots, \tau_{|\pi|-1}$ are the corresponding timestamps from $\mathbb{R}_+$. The signal value at timestamp τ_i is $\sigma_i \in \mathbb{R}^n$. The j-th component of $\sigma_i = \langle a_1, \ldots, a_n \rangle$, namely a_j is denoted $\sigma_i(j)$. In order to simplify presentation, we sometimes assume a timed word to be of the type $(\sigma_0, \tau_0), \ldots (\sigma_{|\pi|-1}, \tau_{|\pi|-1})$. We require the times to be monotonically increasing, that is $\tau_i < \tau_{i+1}$ for all i.

Definition 1 (TSTL Syntax)**.** Given a signal arity n, and a finite set V of freeze-time variables, the formulae of Timed Signal Temporal Logic (TSTL) are defined by the grammar:

$$\varphi := s_k \sim r_k \mid x \sim r \mid \neg\varphi \mid \varphi_1 \substack{\vee \\ \wedge} \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid x.\varphi$$

where $s_k \in \{s_1, \ldots, s_n\}$ is a signal variable, x is a freeze-time variable, $r \in \mathbb{R}_+$, and $r_k \in \mathbb{R}$, and $\sim \in \{<, >, \leq, \geq, =\}$ is the standard comparison operator. $\Box$

The quantifier "$x.$" is known as the *freeze quantifier*, and binds variable x to the current time. We use the shorthand $z \in [a, b]$ to stand for $(z \geq a) \wedge (z \leq b)$.

Definition 2 (Semantics)**.** Let $\pi = (\sigma_0, \tau_0), (\sigma_1, \tau_1), \ldots, (\sigma_{|\pi|-1}, \tau_{|\pi|-1})$ be a finite timed signal of arity n. For a given environment $\mathcal{E} : V \rightarrow \mathbb{R}_+$ binding freeze variables to time values, and a given index position $0 \leq i \leq |\pi| - 1$, the satisfaction relation $(\pi, i, \mathcal{E}) \models \varphi$ for a TSTL formula φ of arity n (with freeze variables in V) is defined as follows.

- $(\pi, i, \mathcal{E}) \models s_k \sim r_k$ iff $\sigma_i(k) \sim r_k$ for signal variable s_k.
- $(\pi, i, \mathcal{E}) \models \neg\varphi$ iff $(\pi, i, \mathcal{E}) \not\models \varphi$.
- $(\pi, i, \mathcal{E}) \models \varphi_1 \substack{\vee \\ \wedge} \varphi_2$ iff $(\pi, i, \mathcal{E}) \models \varphi_1 \substack{\text{or} \\ \text{and}} (\pi, i, \mathcal{E}) \models \varphi_2$.
- $(\pi, i, \mathcal{E}) \models \Diamond\varphi$ iff $\exists j$, $i \leq j < |\pi|$ such that $(\pi, j, \mathcal{E}) \models \varphi$.
- $(\pi, i, \mathcal{E}) \models \Box\varphi$ iff $\forall j$, $i \leq j < |\pi|$, we have $(\pi, j, \mathcal{E}) \models \varphi$.

- $(\pi, i, \mathcal{E}) \models \varphi_1 U \varphi_2$ iff $\exists j$ with $i \leq j \leq |\pi|-1$ s.t. $(\pi, j, \mathcal{E}) \models \varphi_2$ and $\forall k, i \leq k < j$, we have $(\pi, k, \mathcal{E}) \models \varphi_1$.
- $(\pi, i, \mathcal{E}) \models x \sim r$ iff $(\tau_i - \mathcal{E}(x)) \sim r$.
- $(\pi, i, \mathcal{E}) \models x.\varphi$ iff $(\pi, i, \mathcal{E}[x := \tau_i]) \models \varphi$; where $\mathcal{E}[x := \tau_i]$ denotes the environment $\mathcal{E}'$ defined as $\mathcal{E}'(y) = \mathcal{E}(y)$ for $y \neq x$, and $\mathcal{E}'(x) = \tau_i$.

We say the trace π satisfies a TPTL formula φ if $(\pi, 0, \mathcal{E}[\equiv 0]) \models \pi$ where $\mathcal{E}[\equiv 0]$ denotes the freeze variable environment where all variables are mapped to 0. $\square$

Given a signal of arity n, for the i-th timestamp τ_i, we refer to the value of the k-th signal dimension as $s_k(\tau_i)$, which has the value $\sigma_i(k)$.

Definition 3 (Free Variables)**.** The set of free variables $\mathrm{Free}(\varphi)$ in a TSTL formula φ are defined inductively as:

- $\mathrm{Free}(s_k \sim r_k) = \emptyset$, where s_k is a signal variable.
- $\mathrm{Free}(x \sim r) = \{x\}$, where x is a freeze variable.
- $\mathrm{Free}(\varphi_1 \overset{\vee}{\wedge} \varphi_2) = \mathrm{Free}(\varphi_1) \cup \mathrm{Free}(\varphi_2)$.
- $\mathrm{Free}(\neg\varphi) = \mathrm{Free}(\varphi)$; $\quad \mathrm{Free}\left(\overset{\Diamond}{\Box}\varphi\right) = \mathrm{Free}(\varphi)$.
- $\mathrm{Free}(\varphi_1 \,\mathcal{U}\, \varphi_2) = \mathrm{Free}(\varphi_1) \cup \mathrm{Free}(\varphi_2)$.
- $\mathrm{Free}(x.\varphi) = \mathrm{Free}(\varphi) \setminus \{x\}$. $\square$

It can be shown that the environment function $\mathcal{E}$ is only relevant for the free variables in a TSTL formula when it comes to the satisfaction relation $(\pi, i, \mathcal{E}) \models \varphi$. Additionally, if $\mathrm{Free}(\varphi) = \emptyset$, then the environment function is irrelevant in the satisfaction relation $(\pi, i, \mathcal{E}) \models \varphi$.

Example 1. Consider the trace π of arity 2: $((100, 2), 0)$, $((100, 2), 1.1)$, $((100, 5), 3.5)$, $((100, 5), 4.5)$, $((100, 3), 5.1)$, $((100, 2), 5.5)$, $((100, -1), 6.1)$. The last timestamp is 6.1, and the last signal value $\in \mathbb{R}^2$ is $(100, -1)$. The timed word does not satisfy the TSTL formula $\varphi_1 = x.\psi_1 = x.\left((s_1 \geq 50) \to \Diamond\left((s_2 < 0) \wedge (x \leq 2)\right)\right)$. This is because $(\pi, 0, \mathcal{E}[x = 0]) \not\models \psi_1$; as the signal predicate $s_1 \geq 50$ is true at index 0 and there is no subsequent index i such that $\tau_i - 0 \leq 2$ for which $s_2 < 0$ is also true. $\square$

Example 2 (Running example)**.** $\varphi_2 = x.(s_1 \geq 2 \to \Diamond(s_2 > 3 \wedge y.\Diamond(s_3 > 1 \wedge x \leq 5 \wedge y \leq 2)))$. The requirement of φ_2 is: *"If $s_1 \geq 2$ at the start, then $s_2 > 3$ should happen in future, and $s_3 > 1$ should happen in future after $s_2 > 3$, and the duration between $s_2 > 3$ and $s_3 > 1$ should be equal or less than 2 and the duration between $s_1 \geq 2$ and $s_3 > 1$ should be equal or less than 5 ."* In this formula, $\mathrm{Free}(\varphi_2) = \emptyset$. $\square$

III. Quantitative Robustness for TSTL

In this section we define the quantitative semantics for TSTL via a *robustness function* ρ which gives a measure of how well a trace satisfies or violates a given formula. We then explore properties related to our quantitative semantics.

Definition 4 (Quantitative Semantics)**.** Let $\pi = (\sigma_0, \tau_0)$, $(\sigma_1, \tau_1), \ldots, (\sigma_{|\pi|-1}, \tau_{|\pi|-1})$ be a finite timed signal of arity n. For a given environment $\mathcal{E} : V \to \mathbb{R}_+$ binding freeze variables to time values, and a given index position $0 \leq i \leq |\pi|-1$, the robustness function valuation $\rho(\varphi, \pi, i, \mathcal{E}) \in \mathbb{R} \cup \{+\infty, -\infty\}$ for a TSTL formula φ of arity n is defined as

- $\rho(s_k\{\geq, >\}r_k, \pi, i, \mathcal{E}) = \sigma_i(k) - r_k$ for signal variables s_k.
- $\rho(s_k\{\leq, <\}r_k, \pi, i, \mathcal{E}) = r_k - \sigma_i(k)$ for signal variables s_k.
- $\rho(\neg\varphi, \pi, i, \mathcal{E}) = -\rho(\varphi, \pi, i, \mathcal{E})$.
- $\rho(\varphi_1 \wedge \varphi_2, \pi, i, \mathcal{E}) = \min(\rho(\varphi_1, \pi, i, \mathcal{E}), \rho(\varphi_2, \pi, i, \mathcal{E}))$.
- $\rho(\varphi_1 \vee \varphi_2, \pi, i, \mathcal{E}) = \max(\rho(\varphi_1, \pi, i, \mathcal{E}), \rho(\varphi_2, \pi, i, \mathcal{E}))$.
- $\rho(\Box\varphi, \pi, i, \mathcal{E}) = \min\limits_{|\pi| > i' \geq i}(\rho(\varphi, \pi, i', \mathcal{E}))$.
- $\rho(\Diamond\varphi, \pi, i, \mathcal{E}) = \max\limits_{|\pi| > i' \geq i}(\rho(\varphi, \pi, i', \mathcal{E}))$.
- $\rho(\varphi_1 \,\mathcal{U}\, \varphi_2, \pi, i, \mathcal{E}) =$
$$\max_{i' \geq i} \min\left(\rho(\varphi_2, \pi, i', \mathcal{E}), \min_{i'' \in [i, i']} \rho(\varphi_1, \pi, i'', \mathcal{E})\right).$$
- $\rho(x.\varphi, \pi, i, \mathcal{E}) = \rho(\varphi, \pi, i, \mathcal{E}[x := \tau_i])$; where $\mathcal{E}[x := \tau_i]$ denotes the environment $\mathcal{E}'$ defined as $\mathcal{E}'(y) = \mathcal{E}(y)$ for $y \neq x$, and $\mathcal{E}'(x) = \tau_i$.
- $\rho(x \sim r, \pi, i, \mathcal{E}) =$ TRUE if $\tau_i - \mathcal{E}(x) \sim r$; FALSE otherwise for freeze variables x.

For the time constraint formula, $x \sim r$ (the last point in the definition), we consider TRUE $= +\infty$ and FALSE $= -\infty$. If φ has no free variables, we refer to the robustness value of φ over a trace π, denoted $\rho(\varphi, \pi)$, as the value $\rho(\varphi, \pi, 0, \mathcal{E}[\equiv 0])$. $\square$

The time constraint can be seen as an indicator function of the time constraint. We slightly edit the definition of an indicator function, $\mathbf{1}_A(x) =$ TRUE if an element x belongs to the subset A, and $\mathbf{1}_A(x) =$ FALSE if x does not belong to A. Since the TRUE/FALSE values are considered as infinity, it will "restrict" the satisfaction values of the subformulas to be true only inside the interval of the time constraint.

The next lemma will be used to simplify robustness expressions. Consider the cases for the temporal operators in Definition 4; these involve $\max, \min$ over time indices tied to the corresponding temporal operator. The ranges of the time indices depend on the time index "i" in Definition 4. Orthogonally, the time-freeze predicates $x \sim r$ are also present in formulae, and function to restrict the interval in which signal predicates are considered. For example, consider a formula $x.\Diamond\left((x \in [2, 4]) \wedge (s_1 \geq 22) \wedge \theta\right)$. The robustness expression for the formula is $\rho(\varphi, \pi, i = 0, \mathcal{E})$ (assuming $\tau_0 = 0$):

$$\max_{0 \leq j < |\pi|} \min\left((\tau_j \in [2, 4]),\, (s_1(\tau_j) - 22),\, \rho(\theta, \pi, j, \mathcal{E}[x := 0])\right)$$

where we interpret the constraint predicate $(\tau_j \in [2, 4])$ as $+\infty$, *i.e.*, TRUE if the constraint holds, and as $-\infty$, *i.e.*, FALSE otherwise. As the j-th time-stamp is τ_j, we can rewrite the above over the time domain, rather than the time index domain:

$$\max_{0 \leq \tau_j < \tau_{|\pi|}} \min\left((\tau_j \in [2, 4]),\, (s_1(\tau_j) - 22),\, \rho(\theta, \pi, j, \mathcal{E}[x := 0])\right). \tag{1}$$

This is a typical form of robustness expressions. Such expressions can be simplified if the time-freeze constraint $(\tau_j \in [2, 4])$ can be combined with the temporal operator constraint $0 \leq \tau_j < \tau_{|\pi|}$, effectively constrainting the time-range of the temporal operator contraint, reflecting the intention of the time-freeze constraint. The robustness expression in Equation (1) can be seen to be equivalent to:

$$\max_{\tau_j \in [2, 4]} \min\left((s_1(\tau_j) - 22),\, \rho(\theta, \pi, j, \mathcal{E}[x := 0])\right).$$

The next lemma formalizes such simplyfying reductions in the general case. Consider an expression $\max\{\mathrm{constr}(u), \mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u)\}$ of the sort as in Equation (1), where we interpret $\mathrm{constr}(u)$ as $+\infty$, *i.e.*, TRUE if the constraint predicate $\mathrm{constr}(u)$ is true, and $-\infty$, *i.e.*, FALSE otherwise.

Lemma 1. *Consider the expression:*

$$\psi_1 = \max_{u\in I} \min \left(\begin{array}{l} \mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u), \\ \mathrm{constr}(u) \end{array} \right)$$

where, I is a non empty closed bounded subset of $\mathbb{R}$, $\mathrm{expr}_i(u)$ are expressions over u and other variables, with expressions values in $\mathbb{R} \cup \{-\infty, +\infty\}$; and $\mathrm{constr}(u)$ is a constraint predicate over u. The expression ψ_1 is equivalent to

$$\psi_1 = \max_{(u\in I)\wedge \mathrm{constr}(u)} \min\left(\mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u)\right) \tag{2}$$

where if $(u \in I) \wedge \mathrm{constr}(u)$ is not satisfiable, then we interpret $\max_\emptyset()$ as FALSE, i.e.*, $-\infty$. Similarly,*

$$\begin{aligned} \psi_2 &= \min_{u\in I} \max \left(\begin{array}{l} \mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u), \\ \mathrm{constr}(u) \end{array} \right) \\ &= \min_{(u\in I)\wedge \neg\, \mathrm{constr}(u)} \max\left(\mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u)\right) \end{aligned} \tag{3}$$

where if $(u \in I) \wedge \neg \mathrm{constr}(u)$ is not satisfiable, then we interpret $\min_\emptyset()$ as TRUE, i.e.*, $+\infty$.* □

Let $\Theta(u) = \max(\mathrm{expr}_1(u), \ldots, \mathrm{expr}_m(u), \mathrm{constr}(u))$. We have $\psi_2 = \min_{u\in I} \Theta(u)$. If $\mathrm{constr}(u) =$ TRUE, then $\Theta(u) = +\infty$. Hence, Equation 3 follows. Equation 2 results from similar reasoning.

Theorem 1. *Let φ be a* TSTL *formula, π a trace, i a timestamp index, and $\mathcal{E}$ a freeze variable environment.*

1) *If $\rho(\varphi, \pi, i, \mathcal{E}) > 0$ then $(\pi, i, \mathcal{E}) \models \varphi$.*
2) *If $\rho(\varphi, \pi, i, \mathcal{E}) < 0$ then $(\pi, i, \mathcal{E}) \not\models \varphi$.*

If $\rho(\varphi, \pi, i, \mathcal{E}) = 0$, we can not conclude. □

In STL, timing intervals I over temporal operators such as $\Diamond_I$ are not a cause of formula violation by *themselves* so long as $I \neq \emptyset$, and we have a sufficiently long trace. That is, every trace which is long enough will have timestamps that satisfy the timing constraints of I. In TSTL however, one can write formulae for which no trace will satisfy the timing requirements. Consider for example, a formula $\Box\left(x_1.\psi_1 \wedge \Diamond\left(x_2.\psi_2 \wedge \Diamond\left(x_1 \in [1,5] \wedge x_2 \in [7,9] \wedge \psi_3\right)\right)\right)$. *No* trace will satisfy this formula due to the timing constraint requirements. The next lemma formalizes this observation.

Proposition 1. *If a* TSTL *formula φ with no free variables gives an infinite robustness value over a given trace π,* i.e.*, $\rho(\varphi, \pi, 0, \mathcal{E}) =$* TRUE | FALSE*, then φ will give the same infinite robustness value over any other trace π' having the same timestamps,* i.e.*, $\rho(\varphi, \pi', 0, \mathcal{E}) =$* TRUE | FALSE. □

Proof. The way we defined the TSTL semantics makes TRUE/FALSE values appear in the time constraints first (this is true if our TSTL formula φ does not have any concrete TRUE's or FALSE'es). So, if a TSTL formula φ gives a TRUE/FALSE robustness value, the origin of that value will remain no matter what the signal values are in φ. In other words, the time constraint will give the same robustness values for the different timestamps for any signal inputs and the robustness of the time constraints will "overwrite" the robustness of the signal: the TRUE/FALSE values will spread and appear in the final robustness value $\rho(\varphi, \pi, i, \mathcal{E})$ for any π with the same timestamps. The intuition behind this proposition will be easier to see once we introduce the monitoring table later on (Subsection IV-B). □

A result of this proposition is that such formulas, giving TRUE/FALSE robustness values, are not good TSTL formulae and should be avoided because they "ignore" the signal values. Some example templates are of the form $\varphi_1 = \Box(\text{time constraint} \wedge \varphi')$, or $\varphi_2 = \Diamond(\text{time constraint} \vee \varphi')$ where φ' is a TSTL formula and the time constraint is not identically TRUE or FALSE, that is, the time constraint should hold for some timestamps, and not for others. Here $\rho(\varphi_1, \pi, 0, \mathcal{E}) =$ FALSE for any π because our time constraint will not be TRUE for all timestamps. Similarly, $\rho(\varphi_2, \pi, 0, \mathcal{E}) =$ TRUE.

Proposition 1 also gives a procedure for checking whether the timing constraints are good with respect to a time-stamp sequence $\tau_0, \ldots, \tau_\alpha$: one can pick arbitrary but finite signal values at these timestamps, and compute the robustness values for this random trace. If the robustness value is not finite, then the formula time constraints are not satisfiable over this timestamp sequence.

We next present three examples, over traces of length 7, where $\tau_i = i$.

Example 3. $\varphi_3 = x.\Diamond(s_1 \geq 0 \wedge x \in [1,5] \wedge \Box(x \in [1,2] \rightarrow s_2 \geq 0))$. This formula says that in the interval $[1,5]$, we must have a time t such that at that time $s_1 \geq 0$ and additionally if $t \in [1,2]$ when this happens then we must also have $s_2 \geq 0$. The robustness expression of φ_3 is:

$$\rho(\varphi_3, \pi, 0, \mathcal{E}) = \max_{t\in[0,6]} \min \left(\begin{array}{l} s_1(t), \\ t \in [1,5], \\ \min\limits_{t'\in[t,6]} \max \left(\begin{array}{l} t' \notin [1,2], \\ s_2(t') \end{array} \right) \end{array} \right).$$

This can be understood as follows. The outermost $x.$ freezes x to 0, thus future occurrences of x refer to the trace timestamps. First, we have the $\Diamond$ operator which translates to $\max\limits_{t\in[0,6]}$. Then, we have a $\min$ coming from two $\wedge$ operators that we considered as a single operator. This operator will give the $\min$ value between $s_1(t), t \in [1,5]$ and $\varphi_3^1 = \Box(x \in [1,5] \rightarrow s_2 \geq 0)$.

The robustness value of φ_3^1 at t is $\rho(\varphi_3^1, t, \mathcal{E}[x := 0]) = \min\limits_{t'\in[t,6]} \max\left(t' \notin [1,2], s_2(t')\right)$, the $\min\limits_{t'\in[t,6]}$ comes from the $\Box$ operator and $\max$ corresponds to the $\rightarrow$ operator.

Now, we will explain how we get the robustness subexpressions $t \in [1,5]$ and $t' \notin [1,2]$ from the corresponding freeze time constraints $x \in [1,5]$ and $x \in [1,2]$ respectively. We have $\varphi_3 = x.\varphi_3^2$ where $\varphi_3^2 = \Diamond(s_1 \geq 0 \wedge x \in [1,5] \wedge \Box(x \in [1,2] \rightarrow s_2 \geq 0))$. From the Definition 4, we have $\rho(x.\varphi_3^2, \pi, i, \mathcal{E}) = \rho(\varphi_3^2, \pi, i, \mathcal{E}[x := \tau_i])$, as we are interested in timestamp τ_0, we consider $i = 0$. This definition freezes x to 0. Then, we have $\Diamond.\varphi_3^3$ where $\varphi_3^3 = s_1 \geq 0 \wedge x \in [1,5] \wedge \Box(x \in [1,2] \rightarrow s_2 \geq 0)$. This $\Diamond$ will take the maximum value of $\rho(\varphi_3^3, \pi, i, \mathcal{E})$ for all $0 \leq i \leq |\pi| - 1$ (to simplify notation, we considered $t = \tau_i$). From the last point in Definition 4, for each i, $x \in [1,5]$ is written as $\tau_i - \mathcal{E}(x) \in [1,5] \Leftrightarrow t - 0 \in [1,5]$. Similarly, the second time constraint is $t' \notin [1,2]$.

Now, we will simplify the robustness expression of φ_3. In a first step, we apply Lemma 1.

$$\rho(\varphi_3, \pi, 0, \mathcal{E}) = \max_{t\in[0,6]} \min \left(\begin{array}{l} s_1(t), \\ t \in [1,5], \\ \min\limits_{t'\in[t,2]} s_2(t') \end{array} \right).$$

Then, we apply Lemma 1 again and end up with

$$\rho(\varphi_3, \pi, 0, \mathcal{E}) = \max_{t\in[1,5]} \min \left(\begin{array}{l} s_1(t), \\ \min\limits_{t'\in[t,2]} s_2(t') \end{array} \right).$$

For particular signal ranges, further simplification of robustness expressions is often possible. For example, let us assume that the trace signal values have the following ranges for all timestamps: $s_1 \in [-9,-5]$ and $s_2 \in [-1,-3]$.

If we go back to the expression of the robustness that we obtained in the previous analysis, and considering the new constraints that we have on the values of the signals, we have a new simpler expression of the robustness:

$$\rho(\varphi_3, \pi, 0, \mathcal{E}) = \max_{t\in[0,6]} \min\left(s_1(t), t \in [1,5]\right).$$

This is because for all t we have the expression $\min_{t'\in[t,6]} \max\left(t' \notin [1,2], s_2(t')\right)$ to be always more than $s_1(t)$ for the given trace as any s_2 value is more than any s_1 value; and we have an outer min in the original expression in which $s_1(t)$ dominates. This expression can be further simplified using Lemma 1 to $\max_{t\in[1,5]} s_1(t)$. □

Example 4. $\varphi_4 = x.\Box(s_1 \geq 0 \wedge x \in [1,5] \wedge \Diamond(x \in [1,2] \rightarrow s_2 \geq 0))$. This example is quite similar to the first one, we just swapped the temporal operators. The new expression of the robustness is:

$$\rho(\varphi_4, \pi, 0, \mathcal{E}) = \min_{t\in[0,6]} \min \left(\begin{array}{l} s_1(t), \\ t \in [1,5], \\ \max\limits_{t'\in[t,6]} \max \left(\begin{array}{l} t' \notin [1,2], \\ s_2(t') \end{array} \right) \end{array} \right).$$

We obtained this expression of robustness the same way as in example 1, we just swapped the max and min corresponding to the $\Diamond$ and $\Box$. Here, in this example, we can tell that the robustness value of this formula is equal to FALSE. In fact, the time constraint $x \in [1,5]$ will not be always TRUE for all timestamps $\tau_i, i \in [0,6]$, in particular, for $i = 0$. That time constraint with the two $\wedge$ operators inside the $\Box$ will return FALSE. This example is one of the "bad" examples we introduced after lemma 2. □

Example 5. Let us consider Example 2 [Running example]. φ_2 can be written as $x.\psi_2$ where $\psi_2 = s_1 \geq 2 \rightarrow \Diamond(s_2 > 3 \wedge y.\Diamond(s_3 > 1 \wedge x \leq 5 \wedge y \leq 2))$. The robustness expression of φ_2 is: $\rho(\varphi_2, \pi, 0, \mathcal{E}[\equiv 0]) = \rho(\psi_2, \pi, 0, \mathcal{E}[x := 0]) =$

$$\max \left(\begin{array}{l} -(s_1(0)-2), \\ \max\limits_{t\in[0,6]} \min \left(\begin{array}{l} s_2(t)-3, \\ \max\limits_{t'\in[t,6]} \min \left(\begin{array}{l} s_3(t')-1, \\ t' \leq 5, \\ t'-t \leq 2 \end{array} \right) \end{array} \right) \end{array} \right).$$

Now, we analyze this expression. First, we have the max corresponding to the $\rightarrow$ operator. Second, we have a $\Diamond$ operator which is represented by a max over the whole trace. The min afterwards is the $\wedge$ operator. After that we have another $\Diamond$ which covers the remaining trace starting from t. And finally, we have two $\wedge$ represented by a single min.

If we apply Lemma 1, the expression $\rho(\psi_2, \pi, 0, \mathcal{E}[x := 0]) =$ can be simplified to:

$$\max \left(\begin{array}{l} -(s_1(0)-2), \\ \max\limits_{t\in[0,6]} \min \left(\begin{array}{l} s_2(t)-3, \\ \max\limits_{t'\in[t,5]\wedge t'-t\leq 2} s_3(t')-1 \end{array} \right) \end{array} \right). \quad \Box$$

Next, we present the TSTL fragment in which at most one freeze variable is free at any time instant. This fragment is already more expressive than STL, and we show later, admits very efficient algorithms.

Definition 5 (Subformulae)**.** Given a TSTL formula φ, the corresponding subformulae $\mathrm{Sub}(\varphi)$ are defined inductively as:

- $\mathrm{Sub}(s_k \sim r_k) = \{s_k \sim r_k\}$ for signal predicate $s_k \sim r_k$.
- $\mathrm{Sub}(x \sim r) = \{x \sim r\}$ for time constraint $x \sim r$.
- $\mathrm{Sub}(\neg\varphi) = \{\neg\varphi\} \cup \mathrm{Sub}(\varphi)$.
- $\mathrm{Sub}\left(\varphi_1 \overset{\vee}{\wedge} \varphi_2\right) = \{\varphi_1 \overset{\vee}{\wedge} \varphi_2\} \cup \mathrm{Sub}(\varphi_1) \cup \mathrm{Sub}(\varphi_2)$.
- $\mathrm{Sub}(\varphi_1 \,\mathcal{U}\, \varphi_2) = \{\varphi_1 \,\mathcal{U}\, \varphi_2\} \cup \mathrm{Sub}(\varphi_1) \cup \mathrm{Sub}(\varphi_2)$.
- $\mathrm{Sub}\left(\overset{\Diamond}{\Box}\varphi\right) = \left\{\overset{\Diamond}{\Box}\varphi\right\} \cup \mathrm{Sub}(\varphi)$.
- $\mathrm{Sub}(x.\varphi) = \{x.\varphi\} \cup \mathrm{Sub}(\varphi)$. □

Definition 6 (TSTL_1 fragment)**.** A TSTL formula φ is a TSTL_1 formula provided all of the following conditions hold.

1) For every subformula $\psi \in \mathrm{Sub}(\varphi)$, we have $|\mathrm{Free}(\psi)| \leq 1$, *i.e.*, every subformula can have at most one free variable; and
2) Corresponding to every subformula $x.\psi \in \mathrm{Sub}(\varphi)$ involving a freeze quantifier $x.$, we have $\mathrm{Free}(\psi)$ to be either $\emptyset$, or $\{x\}$, that is if $x.\psi$ is a subformula of φ, then ψ cannot have any free variables apart from x.
3) All freeze quantifiers are over unique variables. □

Intuitively, a TSTL_1 formula is one in which only freeze variable is "active" (by being free) in any given subformula. For instance the formula $\Diamond y.(y > 50 \wedge \Box(x.(a \vee x \in [10,20])))$ is a TSTL_1 formula, but $\Diamond y.(y > 50 \wedge \Box(x.(a \vee x \in [10,20] \vee y \in [60,80])))$ is not, as it has the subformula $(a \vee x \in [10,20] \vee y \in [60,80])$ which has two free variables. Also, all the formulas we introduced in the previous examples are TSTL_1 formulas except for φ_4 in Example 2. Every TSTL_1 formula can be transformed into an equivalent TSTL formula with just one freeze variable, with multiple freezing instances.

Proposition 2. *For any* STL *formula* φ

1) *There is an equivalent* TSTL_1 *formula* $\varphi_{\mathrm{TSTL}_1}$.
2) *The* TSTL *robustness value* $\rho(\varphi_{\mathrm{TSTL}_1}, \pi)$ *for* $\varphi_{\mathrm{TSTL}_1}$ *according to Definition 4 gives the same value as the* STL *robustness value* $\rho(\varphi, \pi)$ *for* φ *from [14].* □

IV. Robustness Computation Algorithm for TSTL

A. Syntax Trees

Each TSTL (or TSTL_1) formula has a corresponding syntax tree which depicts the hierarchical syntactic structure of the formula. Our monitoring procedure will depend on this syntax tree.

Definition 7 (Syntax Tree)**.** Given a TSTL formula φ, the associated abstract syntax tree $\mathrm{AST}(\varphi)$ is defined as follows.

- The nodes of the syntax tree are $\mathrm{Sub}(\varphi)$.
- The root node is φ.
- The edges in the tree are defined by the operator structure:
  - If $\neg\psi \in \mathrm{Sub}(\varphi)$, then $\neg\psi$ has the child ψ.
  - If $x.\psi \in \mathrm{Sub}(\varphi)$, then $x.\psi$ has the child ψ.
  - If $\mathsf{op}\,\psi \in \mathrm{Sub}(\varphi)$, for $\mathsf{op} \in \{\bigcirc, \Box, \Diamond\}$, then $\mathsf{op}\,\psi$ has the child ψ.
  - If $\psi_1\,\mathsf{op}\,\psi_2 \in \mathrm{Sub}(\varphi)$, for $\mathsf{op} \in \{\wedge, \vee, \rightarrow, \mathcal{U}\}$, then $\psi_1\,\mathsf{op}\,\psi_2$ has the two children ψ_1, ψ_2. □

In order to check whether a timed word satisfies a TSTL_1 formula φ, we build on the insight of [12] which noted that we can compute the satisfaction relation for subformulae involving only one free variable (for various word position indices), and after this computation, use these values akin to values computed had the subformula been an STL formula (without any freeze variables). The basic structure over which our algorithm will operate will be subtrees corresponding to various freeze variables.

Definition 8 (Sub-trees for Freeze Variables)**.** Let φ be a TSTL_1 formula with the freeze variable set $V \neq \emptyset$. Consider the reverse topological sort of the nodes of its syntax tree $\text{AST}(\varphi)$, and consider the ordering of the nodes of the form $x.\psi$ in this sort. Let $x_1, \ldots, x_{|V|}$ be an ordering of the freeze variables in φ consistent with the variable ordering indicated by $x.\psi$ in the reverse topological sort. We define subtrees $\text{SubTree}_\varphi(x_j)$ for $x_j \in V$ in a bottom up fashion as follows. Let $\text{AST}(\varphi, \theta)$ denote the subtree of $\text{AST}(\varphi)$ rooted at θ for $\theta \in \text{Sub}(\varphi)$.

- $\text{SubTree}_\varphi(x_1)$ is the subtree $\text{AST}(\varphi, \psi_1)$ where $x_1.\psi_1$ is a subformula of φ corresponding to the freeze variable x_1. Note that in our formulae, each freeze operator must correspond to a unique freeze variable.
- $\text{SubTree}_\varphi(x_j)$ for $j > 1$ is the subgraph (this subgraph can be shown to be a subtree):
$$\text{AST}(\varphi, \psi_j) \setminus \left(\bigcup_{1 \le i < j} \text{SubTree}_\varphi(x_i) \right)$$
where $x_j.\psi_j$ is the subformula of φ corresponding to the freeze variable x_j.

We also have a subtree at the "top" which need not correspond to any freeze variable, *e.g.*, if the root node of $\text{AST}(\varphi)$ is not a freeze operator node. We call this subtree the top subtree, $\text{TopSubTree}(\varphi)$, defined as
$$\text{TopSubTree}(\varphi) = \text{AST}(\varphi) \setminus \left(\bigcup_{1 \le i \le |V|} \text{SubTree}_\varphi(x_i) \right).$$

Each $\text{SubTree}_\varphi(x_j)$ which is not a TopSubTree has a parent and a root: (a) the root node, denoted as $\text{SubTree}_\varphi(x_j).\mathit{root}$, is the node for which the parent in $\text{AST}(\varphi)$ does not belong to $\text{SubTree}_\varphi(x_j)$; (b) the parent node $\text{SubTree}_\varphi(x_j).\mathit{parent}$ of the subtree $\text{SubTree}_\varphi(x_j)$ is the parent of node $\text{SubTree}_\varphi(x_j).\mathit{root}$ in $\text{AST}(\varphi)$. $\square$

In the above definition, we note that if the root node of $\text{AST}(\varphi)$ is a freeze operator node $x.\psi$, the top subtree, $\text{TopSubTree}(\varphi)$ has only one node: $x.\psi$.

Example 6. Consider the formula from Example 2 ([Running example]), and its associated syntax tree in Figure 1. The formula subscripts correspond to a reverse topological sort of the syntax tree. The freeze variable ordering given by a reverse topological sort is $y <_{\mathsf{revtop}} x$. The nodes in $\text{SubTree}_\varphi(y)$ are $\{\varphi_8, \varphi_9, \varphi_{10}, \varphi_{11}, \varphi_{12}\}$. The nodes in $\text{SubTree}_\varphi(x)$ are $\{\varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7\}$; and in $\text{TopSubTree}_\varphi$ are $\{\varphi_1\}$. Also, $\text{SubTree}_\varphi(y).\mathit{root} = \varphi_8$, $\text{SubTree}_\varphi(y).\mathit{parent} = \varphi_7$, $\text{SubTree}_\varphi(x).\mathit{root} = \varphi_2$ and $\text{SubTree}_\varphi(x).\mathit{parent} = \varphi_1$. $\square$

B. Monitoring Table

The monitoring table of a TSTL formula φ is a table that has the subformulas of ϕ as its rows as showing in the syntax tree $\text{AST}(\varphi)$, and the time stamps τ_i for $i \in [0, |\pi| - 1]$ of a trace π as its columns. We use the algorithms below to fill the monitoring table in order to get the robustness of φ. Tables I and II correspond to φ_2 in Example 2 [Running example].

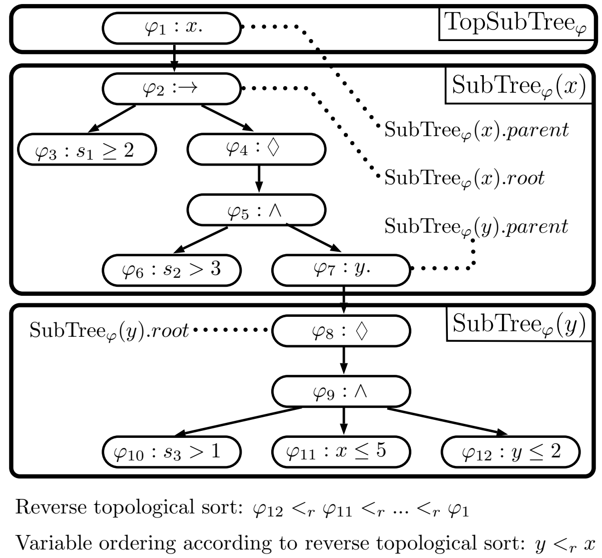

Fig. 1: Syntax Tree for Example 2 [Running example].

C. Algorithm Overview

The main level algorithm TSTL Monitor Algorithm 2 computes the robustness by calling the recursive procedure Rec-TSTL in Algorithm 3. This procedure Rec-TSTL $(1, 0)$ essentially considers all possible freeze bindings $\tau_0 = x_1 \leq x_2 \leq \ldots x_n \leq \tau_{|\pi|-1}$ via its recursive call chain, and for all such bindings it computes the values of all the subformulae in the syntax tree by repeated calls to a sub-procedure ComputeRobustness 1. ComputeRobustness in turn computes all the robustness values of the subformulae in a subtree $\text{SubTree}_\varphi(x_k)$ in a bottom up fashion, for each freeze binding environment provided the robustness values for all subformulae $y.\psi_y$ have been computed for all freeze variables y such that $y <_{\mathsf{revtop}} x$, where $<_{\mathsf{revtop}}$ denotes a reverse topological ordering of the freeze variables in the syntax tree for φ. Rec-TSTL calls also ensure that when the root node robustness for a freeze variable subtree $\text{SubTree}_\varphi(x_k)$ has been computed, it is copied over to its parent subtree $\text{SubTree}_\varphi(x_{k-1})$ appropriately.

For a freeze variable order $x_{|V|} < \ldots < x_k \ldots < x_1$, at a particular point k in the call chain: (1) the algorithm *fixes* the values of the freeze variables x_1 through x_k; then (2) computes the values of all subtrees $\text{SubTree}_\varphi(x_{k+1})$ through $\text{SubTree}_\varphi(x_{|V|})$ *for all possible* values of x_{k+1} through $x_{|V|}$; next (3) computes the values of all the nodes in $\text{SubTree}_\varphi(x_k)$ in a bottom up fashion which can now be done since we have the values of all subtrees $\text{SubTree}_\varphi(x_{k+1})$ through $\text{SubTree}_\varphi(x_{|V|})$ for all possible values of x_{k+1} through $x_{|V|}$. It then picks different values for x_1 through x_k and repeats the process.

D. Algorithm Details

1) ComputeRobustness (Algorithm 1): ComputeRobustness computes the robustness values of the TSTL subformulas in a subtree $\text{SubTree}_\varphi(x_i)$ provided the robustness values of the leaves have been precomputed. It does this by filling up table entries from the last timestamp in a backwards fashion as in the LTL monitoring algorithm of [36]. The interesting case is the until operator which can be understood as follows. $(\pi, i, \mathcal{E}) \models \psi_1 \,\mathcal{U}\, \psi_2$ iff either (a) $(\pi, i, \mathcal{E}) \models \psi_2$; or (b) $(\pi, i, \mathcal{E}) \models \psi_1$, and additionally $(\pi, i+1, \mathcal{E}) \models \psi_1 \,\mathcal{U}\, \psi_2$ which translates to $\rho(\psi_1 \,\mathcal{U}\, \psi_2, \pi, i, \mathcal{E}) = \max(\rho(\psi_2, \pi, i, \mathcal{E}), \min(\rho(\psi_1, \pi, i, \mathcal{E}), \rho(\psi_1 \,\mathcal{U}\, \psi_2, \pi, i + 1, \mathcal{E})))$ in the robustness world.

Algorithm 1: ComputeRobustness

Input: $\varphi_j, u, M_{|\varphi| \times |\pi|}$
Output: $M_{|\varphi| \times |\pi|}$ // Only entry $M[j, u]$ is changed.
if $\varphi_j \equiv \neg\varphi_m$ **then return** $-M[m, u]$
else if $\varphi_j \equiv \varphi_m \wedge \varphi_n$ **then return** $\min(M[m, u], M[n, u])$
else if $\varphi_j \equiv \varphi_m \vee \varphi_n$ **then return** $\max(M[m, u], M[n, u])$
else if $\varphi_j \equiv \Box\varphi_m$ **then**
  if $u = |\pi| - 1$ **then return** $M[m, u]$
  else return $\min(M[m, u], M[j, u + 1])$
else if $\varphi_j \equiv \Diamond\varphi_m$ **then**
  if $u = |\pi| - 1$ **then return** $M[m, u]$
  else return $\max(M[m, u], M[j, u + 1])$
else if $\varphi_j \equiv \varphi_m U \varphi_n$ **then**
  if $u = |\pi| - 1$ **then return** $M[n, u]$
  else return $\max(M[n, u], \min(M[m, u], M[j, u + 1]))$
else if $\varphi_j \equiv x_k \sim r$ *OR* $\varphi_j \equiv s \sim r$ **then**
  return $M[j, u]$ `// Already computed by calling function`

2) TSTL Monitor Algorithm (Algorithm 2): The first line of the TSTL Monitor Algorithm calculates the values of the signal predicates $s\{\leq, <, \geq, >\}r$ for the different timestamps τ_i for $i \in [0, |\pi| - 1]$. The second line calls the algorithm Rec-TSTL to calculate the values of $\mathrm{SubTree}_\varphi(x_k)$ for every freeze variable x_k in φ. The remaining lines in the algorithm (3 to 6) calculate the values of $\mathrm{TopSubTree}(\varphi)$, this is the case when all the subformulae of type $x.\varphi$ have been computed. Finally, the algorithm returns $M[1, 0]$ which indicates the robustness of the TSTL formula φ at 0. Note that if $|\mathrm{TopSubTree}(\varphi)| = 1$, then the top subtree has only one node $x.\psi$, and the robustness value of ψ for the environment $\mathcal{E}[x := 0]$ has already been computed by Rec-TSTL so there is nothing to do.

Algorithm 2: TSTL Monitor Algorithm

Input: $\varphi, \pi = (\sigma_0, \tau_0), \ldots, (\sigma_T, \tau_T)$, Θ = Syntax Tree for φ; **Global Table:** $M_{|\varphi| \times |\pi|}$
Output: $M[1, 0]$.
1 Initialize all rows in $M_{|\varphi| \times |\pi|}$ corresponding to predicates $\varphi_j \equiv s \sim r$ with real values according to $\forall 0 \leq i \leq |\pi| - 1, M[j, i] = \mp(s(\tau_i) - r)$
2 Rec-TSTL(1,0)
3 **if** $|\mathrm{TopSubTree}(\varphi)| \geq 2$ **then**
4   **for** $i \leftarrow |\pi| - 1$ *down to* 0 **do**
5   **for** $j \leftarrow \mathrm{TopSubTree}(\varphi).\max$ *down to* $\mathrm{TopSubTree}(\varphi).\min$ **do**
6       $M[j, i] \leftarrow \mathit{ComputeRobustness}(\varphi_j, i, M_{|\varphi| \times |\pi|})$
7 **return** $M[1, 0]$

3) $\mathit{Rec\text{-}TSTL}(k, t)$ overview (Algorithm 3): This function $\mathit{Rec\text{-}TSTL}(k, t)$ calculates the values of all the sub-trees $\mathrm{SubTree}_\varphi(x_j)$, $j \geq k$ for the different instantiations of x_k to τ_i for $i \in [t, |\pi| - 1]$.

4) Technical details: $\mathit{Rec\text{-}TSTL}(k, t)$: This subsection can be omitted on the first reading. The pre-condition for a function call $\mathit{Rec\text{-}TSTL}(k, t)$ is that all the time-constraints $x_l \sim r$ for $l < k$ have already been computed for x_l instantiated to τ_t and assigned to the appropriate table locations in M. Before going through a technical explanation, let us give an intuitive idea on how the algorithm works. $\mathit{Rec\text{-}TSTL}(1, 0)$ is called in line 2 of Algorithm 2. This call will result in the calculation of $\rho(\varphi, \pi, 0, \mathcal{E})$. Let us take an example and suppose our TSTL formula φ is of the form $\varphi = ...x_1.(...x_2.(...x_3.(...)))$ where the dots can be any operators and φ has 3 freeze time variables. The first call $\mathit{Rec\text{-}TSTL}(1, 0)$ will calculate the time constraints corresponding to x_1 for the instantiation τ_0. Then, $\mathit{Rec\text{-}TSTL}(1, 0)$ will call $\mathit{Rec\text{-}TSTL}(2, 0)$ to calculate the time constraints corresponding to x_2 for the instantiation τ_0, and afterwards, $\mathit{Rec\text{-}TSTL}(3, 0)$ is called to calculate the time constraints corresponding to x_3 for the instantiation τ_0. Since x_3 is the last freeze variable in φ, $\mathit{Rec\text{-}TSTL}(3, 0)$ will continue to calculate $\mathrm{SubTree}_\varphi(x_3)$ for all the instantiation of x_3 to τ_i for $i \in [0, |\pi| - 1]$. Once that is done, we go back to $\mathit{Rec\text{-}TSTL}(2, 0)$ to calculate $\mathrm{SubTree}_\varphi(x_2)$ corresponding to x_2 instantiated to τ_0. Then, for $i = 1$ in $\mathit{Rec\text{-}TSTL}(2, 0)$, we calculate the time constraints corresponding to x_2 for the instantiation τ_1 and call $\mathit{Rec\text{-}TSTL}(3, 1)$ which will calculate $\mathrm{SubTree}_\varphi(x_3)$ again but this time for all instantiations of x_3 starting from τ_1 and so on...

This is how $\mathit{Rec\text{-}TSTL}$ is called : $\mathit{Rec\text{-}TSTL}(1, 0) \rightarrow \mathit{Rec\text{-}TSTL}(2, 0) \rightarrow \mathit{Rec\text{-}TSTL}(3, 0) \rightarrow \mathit{Rec\text{-}TSTL}(3, 1) \ldots \rightarrow \mathit{Rec\text{-}TSTL}(3, |\pi| - 1) \rightarrow \mathit{Rec\text{-}TSTL}(2, 1) \rightarrow \mathit{Rec\text{-}TSTL}(3, 1) \rightarrow \mathit{Rec\text{-}TSTL}(3, 2) \ldots \rightarrow \mathit{Rec\text{-}TSTL}(3, |\pi| - 1) \ldots \rightarrow \mathit{Rec\text{-}TSTL}(2, |\pi| - 1) \rightarrow \mathit{Rec\text{-}TSTL}(3, |\pi| - 1)$.

In general, for the instantiation of x_k to $\tau_i = \tau_t$ (Line 1), we calculate the values of the time constraints $\varphi_j = x_k \sim r$ for all timestamps τ_u for $u \in [i, |\pi| - 1]$. Then, in a recursive way, the algorithm calculates the values of the time constraints $\varphi_j = x_{k'} \sim r$ for all the remaining freeze variables $x_{k'}, k \leq k' \leq |V|$ instantiated to τ_i (Line 6). Once the algorithm reaches the final freeze variable $x_{|V|}$, we already have all the time constraints of the different freeze variables calculated corresponding to all freeze variable instantiated to τ_i, and the algorithm calculates the values of $\mathrm{SubTree}_\varphi(x_{|V|})$ for the different timestamps $\tau_u, u \in [i, |\pi| - 1]$ (Lines 7-9) and copies $M[\mathrm{SubTree}_\varphi(x_{|V|}).\mathit{root}, i]$ to $M[\mathrm{SubTree}_\varphi(x_{|V|}).\mathit{parent}, i]$. Then, it instantiates $x_{|V|}$ to the next timestamp, calculates $\mathrm{SubTree}_\varphi(x_{|V|})$ for the different timestamps $\tau_u, u \in [i + 1, |\pi| - 1]$ and copies $M[\mathrm{SubTree}_\varphi(x_{|V|}).\mathit{root}, i + 1]$ to $M[\mathrm{SubTree}_\varphi(x_{|V|}).\mathit{parent}, i + 1]$ and so on until we finish with all the instantiations of $x_{|V|}$ (Line 1). Once that is done, we go back to the previous call of $\mathit{Rec\text{-}TSTL}(|V| - 1, i)$ and calculate the values of $\mathrm{SubTree}_\varphi(x_{|V|-1})$ for the different timestamps $\tau_u, u \in [i, |\pi| - 1]$. Then, $x_{|V|-1}$ is instantiated to $i+1$ and we call $\mathit{Rec\text{-}TSTL}(|V|, i+1)$ and so on. While it may look complicated, the idea behind the $\mathit{Rec\text{-}TSTL}$ algorithm is simple: in order to calculate $\mathrm{SubTree}_\varphi(x_k)$ at the instantiation i, we calculate $\mathrm{SubTree}_\varphi(x_{k+1})$ at the instantiations i', $i' \in [i, |\pi| - 1]$.

We run TSTL Monitor Algorithm on φ_4 from Example 2, Table I shows the monitoring table of φ_4 after the first instantiation of x, *i.e.*, for $x = 0$ after all the instantiations of y have finished (for $x = 0$). Table II shows the final snapshot of the monitoring table. The values of the robustness monitoring table will always have the signal predicate robustness values

Algorithm 3: *Rec-TSTL*(k, t)

for $i \leftarrow t$ *to* $|\pi| - 1$ **do**
  for $u \leftarrow i$ *to* $|\pi| - 1$ **do**
    for *each* $\varphi_j = x_k \sim r$ **do**
      if $\tau_u - \tau_i \sim r$ **then** $M[j, u] \leftarrow$ TRUE;
      else $M[j, u] \leftarrow$ FALSE;
  if $k < |V|$ **then** Rec-TSTL $(k + 1, i)$;
  for $u \leftarrow |\pi| - 1$ *down to* i **do**
    for $j \leftarrow \mathrm{SubTree}_\varphi(x_k).\max$ *down to* $\mathrm{SubTree}_\varphi(x_k).\min$ **do**
      $M[j, u] \leftarrow ComputeRobustness(\varphi_j, u, M_{|\varphi| \times |\pi|})$
  $M[\mathrm{SubTree}_\varphi(x_k).parent, i] \leftarrow M[\mathrm{SubTree}_\varphi(x_k).root, i]$

$(\mp(s(\tau_i) - r))$, or TRUE/FALSE (due to the freeze variable constraints). Since the only operators used to fill up new table entries are $\max / \min$, no new values will be generated.

ψ_i \ τ_i	0	1	2	3	4	5	6
$\varphi_2 = x.\psi_2$	5						
$\psi_2 = \psi_3 \rightarrow \psi_4$	5	5	5	6	8	-2	-9
$\psi_3 = s_1 \geq 2$	3	5	1	-6	-8	2	9
$\psi_4 = \Diamond\psi_5$	5	5	5	-1	-2	-4	F
$\psi_5 = \psi_6 \wedge \psi_7$	-3	-4	5	-1	-2	-4	F
$\psi_6 = s_2 > 3$	-3	4	5	-1	-2	-4	2
$\psi_7 = y.\psi_8$	1	-4	7	7	7	4	F
$\psi_8 = \Diamond\psi_9$	1	-4	7	7	7	4	F
$\psi_9 = \psi_{10} \wedge \psi_{11} \wedge \psi_{12}$	1	-5	-5	-4	7	4	F
$\psi_{10} = s_3 > 1$	1	-5	-5	-4	7	4	8
$\psi_{11} = x \leq 5$	T	T	T	T	T	T	F
$\psi_{12} = y \leq 2$	T	T	T	T	T	T	T

TABLE I: Monitoring table of φ_2 [Running example] after first instantiation of x.

ψ_i \ τ_i	0	1	2	3	4	5	6
$\varphi_2 = x.\psi_2$	5	5	5	6	8	2	2
$\psi_2 = \psi_3 \rightarrow \psi_4$	5	5	5	6	8	2	2
$\psi_3 = s_1 \geq 2$	3	5	1	-6	-8	2	9
$\psi_4 = \Diamond\psi_5$	5	5	5	2	2	2	2
$\psi_5 = \psi_6 \wedge \psi_7$	-3	-4	5	-1	-2	-4	2
$\psi_6 = s_2 > 3$	-3	4	5	-1	-2	-4	2
$\psi_7 = y.\psi_8$	1	-4	7	7	8	8	8
$\psi_8 = \Diamond\psi_9$	1	-4	7	7	8	8	8
$\psi_9 = \psi_{10} \wedge \psi_{11} \wedge \psi_{12}$	1	-5	-5	-4	7	4	8
$\psi_{10} = s_3 > 1$	1	-5	-5	-4	7	4	8
$\psi_{11} = x \leq 5$	T	T	T	T	T	T	T
$\psi_{12} = y \leq 2$	T	T	T	T	T	T	T

TABLE II: Final snapshot of the monitoring table of φ_2 [Running example].

V. Linear Time Robustness Computation Algorithm for TSTL_1

The complexity of the robustness computation procedure in Section IV is exponential in the number of freeze variables (the exponent being over the trace length). A careful analysis shows that in the case of just one freeze variable, *i.e.*, for formulae from TSTL_1, the running time is quadratic in the length of the trace. In this section, for the special case of TSTL_1 formulae, we develop an optimized robustness computation procedure which runs in time *linear* in the trace length.

A. Algorithm Overview

The general idea is the same as in Section IV. The main level procedure is the TSTL_1 Monitor Algorithm 4 which considers all possible freeze bindings for the single freeze variable present in any subtree $\mathrm{SubTree}_\varphi(x_k)$, computes the robustness values for all subformulae in this subtree in a bottom up fashion and then moves to the parent subtree. We improve the complexity to be *linear* from quadratic in the trace length, by tightening the coupling between freeze variable bindings, and subsequent LTL type computation phases. We show that for each *subsequent* binding for x, after the satisfaction relation has been computed for the first binding $x = \tau_0$ the LTL type computation can be done in time independent of the word length by carefully keeping track of what timing constraint predicates and hence robustness values for subformulae can change in going from $x = \tau_i$ to $x = \tau_{i+1}$, and reusing stored robustness values from $x = \tau_i$. This result is made possible by the fact that TSTL only allows base freeze variable constraints of the from $x \sim r$ where $\sim \in \{\leq, \geq, <, >, =\}$ which behave in a monotonic fashion with increasing timestamp values.

B. TSTL_1 Monitor Algorithm (Algorithm 4)

Data Structures: The main data structures used in the TSTL_1 monitor algorithm are:
(a) A $\mathrm{SubTree}_\varphi(x_k)$ 8 for each freeze time variable x_k.
(b) The monitoring table as described in section IV-B.
(c) A vector $\overline{\alpha}$ of $|\mathrm{AST}(\varphi)|$ items, where each item α_j is the position of the timestamp where the time constraint φ_j changed its value from TRUE to FALSE or the opposite (since we only have a low number of time constraints compared to the size of $\mathrm{AST}(\varphi)$, most of the items of α are actually not used). The values of $\overline{\alpha}$ depend on instantiations of the freeze variables to the various timestamps.
(d) Two vectors $\overline{c}$ and $\overline{d}$ of $|\mathrm{AST}(\varphi)|$ items each, where d_j is the position of the largest timestamp where subformula φ_j changed its values from TRUE to FALSE (or the opposite) and c_j is the position of the smallest timestamp where subformula φ_j changed its values from TRUE to FALSE (or the opposite).
(e) An integer t that represents the time index corresponding to the current environment $\mathcal{E}(x_k := \tau_t)$.

Algorithm Logic: Figure 2 gives an overview of the algorithm steps (lines 18 to 22 are not included). Now. we will go over the main steps of TSTL_1 monitor algorithm. For each freeze time variable x_k (line 3), our main algorithm can be split into 3 main phases:
Phase (i). Lines 4-7: this part corresponds to the first instantiation of the freeze variable x_k. This corresponds to box numbers 1 and 2 in Figure 2. Initially, the monitoring table is empty and contains only the values of the signal predicates $s \sim r$ for $\sim \in \{\leq, <, \geq, >\}$ for the different timestamps (line 2). Line 4 calculates the values of the signals for each timestamp corresponding to the first instantiation. Then, lines 5-7 will calculate the values of the subformulas of the subtree $\mathrm{SubTree}_\varphi(x_k)$ for all the timestamps by calling ComputeRobustness (Algorithm 1).

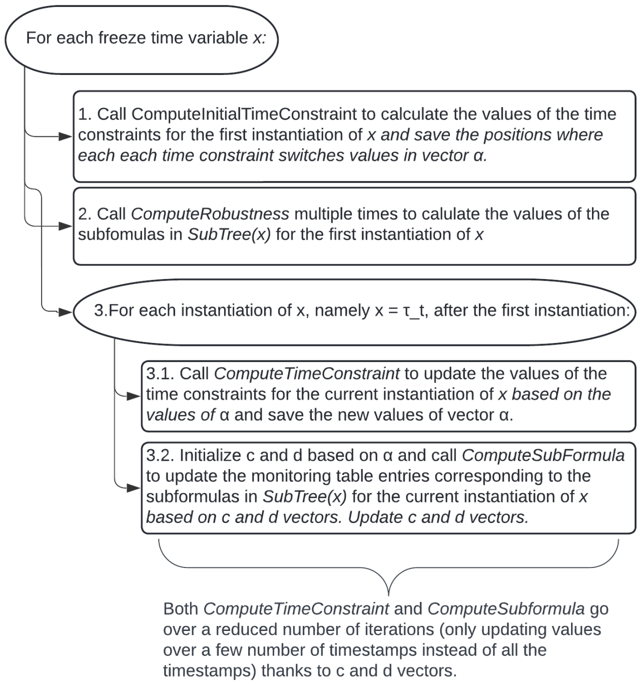

Fig. 2: Algorithm overview

Phase (ii). Lines 9-17. This corresponds to box numbers 3.1 and 3.2 in Figure 2. Here, the $while$ loop in line 9 corresponds to the different instantiations of x_k starting from the 2^{nd} one. Once the condition in this $while$ loop is no longer satisfied, the algorithm stops with the instantiations of x_k: there is no point in doing the calculations for the remaining instantiations since the time constraints values will no longer update thus no changes will be made in the whole $\mathrm{SubTree}_\varphi(x_k)$ (the first position in the vector α where a time constraint changes its value is out of the trace). Based on the instantiation of x_k to τ_t, Line 11 updates the time constraints and the loop in lines 13-14 updates the subformulas in the subtree $\mathrm{SubTree}_\varphi(x_k)$. Variables $\overline{c}$ and $\overline{d}$ are used as inputs for ComputeSubFormula 5 in order to reduce the number of values updated for each subformula (Instead of calculating the values of each subformula for all the timestamps, ComputeSubFormula only updates the values of that subformula at a reduced number of timestamps. Further details can be found later on in section V-G3. Then, t is incremented for the next instantiation.

The loop in lines 16-17 copies the values of $\mathrm{SubTree}_\varphi(x_k).root$ to $\mathrm{SubTree}_\varphi(x_k).parent$ so that it can be used in $\mathrm{SubTree}_\varphi(x_{k+1})$.

Phase (iii). Lines 18-21: this part (not depicted in Figure 2) calculates the values of $\mathrm{TopSubTree}(\varphi)$.

And Finally, the algorithm returns $M[1,0]$ which indicates the robustness of our TSTL_1 formula at time 0.

C. *ComputeInitialTimeConstraint (Algorithm 6)*

ComputeInitialTimeConstraint$(\varphi, x_k, \pi, M_{|\varphi|\times|\pi|})$ corresponds to the first instantiation of the freeze variable x_k to 0. It evaluates the whole time constraints rows (for each τ_i for $i \in [0, |\pi|-1]$) into TRUE/FALSE. In fact, for each time constraint, the algorithm evaluates the corresponding row depending on the operator. More details can be found in Section V-G1.

Algorithm 4: TSTL_1 Monitor Algorithm

Input: $\varphi, \pi = (\sigma_0, \tau_0), \ldots, (\sigma_{|\pi|-1}, \tau_{|\pi|-1})$, Θ = Syntax Tree for φ;
Global Table: $M_{|\varphi|\times|\pi|}$
Output: $M[1,0]$.
2 Initialize all rows in $M_{|\varphi|\times|\pi|}$ corresponding to predicates $\varphi_j \equiv s \sim r$ with real values according to $\forall 0 \le i \le |\pi|, M[j,i] = \mp(s(\tau_i) - r)$
3 **for** $k \leftarrow 1$ *to* $|V|$ **do** // $|V|$: number of freeze variables
4 $[\overline{\alpha}, M] \leftarrow ComputeInitialTimeConstraint(\varphi, x_k, \pi, M_{|\varphi|\times|\pi|})$
5 **for** $i \leftarrow |\pi|-1$ *down to* 0 **do**
6 **for** $j \leftarrow \mathrm{SubTree}_\varphi(x_k).\max$ *down to* $\mathrm{SubTree}_\varphi(x_k).\min$ **do** // $\mathrm{SubTree}_\varphi(x_k).\max$ ($\mathrm{SubTree}_\varphi(x_k).\min$) is the maximum (minimum) index of subformulas in the subtree $\mathrm{SubTree}_\varphi(x_k)$
7 $M[j,i] \leftarrow ComputeRobustness(\varphi_j, i, M_{|\varphi|\times|\pi|})$
8 $t \leftarrow 1$
9 **while** $\min(\overline{\alpha}) < |\pi|$ **do**
10 $\overline{c} \leftarrow \overline{\alpha}$
11 $[\overline{\alpha}, M] \leftarrow ComputeTimeConstraint(\varphi, x_k, \pi, t, M_{|\varphi|\times|\pi|}, \overline{\alpha})$
12 $\overline{d} \leftarrow \overline{\alpha} - 1$
13 **for** $j \leftarrow \mathrm{SubTree}_\varphi(x_k).\max$ *down to* $\mathrm{SubTree}_\varphi(x_k).\min$ **do**
14 $[\overline{c}, \overline{d}, M] \leftarrow ComputeSubFormula(\varphi_j, \overline{c}, \overline{d}, t, M_{|\varphi|\times|\pi|})$
15 $t \leftarrow t + 1$
16 **for** $i \leftarrow 0$ *to* $|\pi|-1$ **do**
17 $M[\mathrm{SubTree}_\varphi(x_k).parent, i] \leftarrow M[\mathrm{SubTree}_\varphi(x_k).root, i]$
18 **if** $|\mathrm{TopSubTree}(\varphi)| \ge 2$ **then**
19 **for** $i \leftarrow |\pi|-1$ *down to* 0 **do**
20 **for** $j \leftarrow \mathrm{TopSubTree}(\varphi).\max$ *down to* $\mathrm{TopSubTree}(\varphi).\min$ **do**
21 $M[j,i] \leftarrow ComputeRobustness(\varphi_j, i, M_{|\varphi|\times|\pi|})$
22 **return** $M[1,0]$

D. *ComputeTimeConstraint (Algorithm 7)*

ComputeTimeConstraint$(\varphi, x_k, \pi, t, M_{|\varphi|\times|\pi|}, \overline{\alpha})$ evaluates the values of time constraints rows for each τ_i for $i \in [t, |\pi|-1])$ where t represents the timestamp index of the current environment for x_k, namely $\mathcal{E}(x_k := \tau_t)$. In fact, it only recalculates the values that changed from the previous instantiation of x_k for a given time constraint and returns the new value of $\overline{\alpha}$. More details can be found in Section V-G2.

E. *ComputeSubFormula (Algorithm 5)*

ComputeSubFormula$(\varphi_j, \overline{c}, \overline{d}, t, M_{|\varphi|\times|\pi|})$ ensures $M[j,i]$ has the correct robustness value $\rho(\varphi_j, \pi, i, \mathcal{E})$ for i in range $[t, |\pi|-1]$. In order to do this, it leverages the work done in ComputeSubFormula$(\varphi_j, \overline{c}, \overline{d}, t-1, M_{|\varphi|\times|\pi|})$ for φ_j in the previous instantiation of x, i.e., for the environment $\mathcal{E}[x := \tau_{t-1}]$. In the worst case, it may need to *update* the $M[j,i]$ table entries (that contained values based on the previous instantiations $(\pi, i, \mathcal{E}[x := \tau_{t-1}]) \models \varphi_j$ from index τ_t till τ_{d_j} (we are guaranteed that the robustness values of φ_j for the timestamps τ_i for $i \in [\tau_{d_j}, \tau_{|\pi|-1}]$ will not change). However, in practice fewer entries in the table need to be updated since we use $while$ loops in ComputeSubFormula (lines 14 and 21) as an early exit condition. More details can be found in Section V-G3.

Algorithm 5: ComputeSubFormula

Input: $\varphi_j, t, \overline{c}, \overline{d}, M_{|\varphi|\times|\pi|}$
Output: $\overline{c}, \overline{d}, M_{|\varphi|\times|\pi|}$
switch φ_j **do**
  case $\neg\varphi_m$ **do**
    $d_j \leftarrow d_m$; $c_j \leftarrow c_m$
    for $k \leftarrow d_j$ *down to* c_j **do**
      $M[k, j] \leftarrow ComputeRobustness(\varphi_j, k, M_{|\varphi|\times|\pi|})$
  case $\varphi_m \wedge \varphi_n, \varphi_m \vee \varphi_n$ **do**
    $d_j \leftarrow \max(d_m, d_n)$; $c_j \leftarrow \min(c_m, c_n)$
    for $k \leftarrow d_j$ *down to* c_j **do**
      $M[k, j] \leftarrow ComputeRobustness(\varphi_j, k, M_{|\varphi|\times|\pi|})$
  case $\Box\varphi_m, \Diamond\varphi_m$ **do**
    $d_j \leftarrow d_m$; $c_j \leftarrow c_m$
    for $k \leftarrow d_j$ *down to* c_j **do**
      $M[j, k] \leftarrow ComputeRobustness(\varphi_j, k, M_{|\varphi|\times|\pi|})$
    while $(M[j, c_j - 1] \neq ComputeRobustness(\varphi_j, c_j - 1, M_{|\varphi|\times|\pi|})$ *AND* $c_j - 1 \geq t)$ **do**
      $M[j, c_j - 1] \leftarrow ComputeRobustness(\varphi_j, c_j - 1, M_{|\varphi|\times|\pi|})$
      $c_j \leftarrow c_j - 1$
  case $\varphi_m \,\mathcal{U}\, \varphi_n$ **do**
    $d_j \leftarrow \max(d_m, d_n)$; $c_j \leftarrow \min(c_m, c_n)$
    for $k \leftarrow d_j$ *down to* c_j **do**
      $M[j, k] \leftarrow ComputeRobustness(\varphi_j, k, M_{|\varphi|\times|\pi|})$
    while $(M[j, c_j - 1] \neq ComputeRobustness(\varphi_j, c_j - 1, M_{|\varphi|\times|\pi|})$ *AND* $c_j - 1 \geq t)$ **do**
      $M[j, c_j - 1] \leftarrow ComputeRobustness(\varphi_j, c_j - 1, M_{|\varphi|\times|\pi|})$
      $c_j \leftarrow c_j - 1$
otherwise do nothing /* leaf nodes of subtree; already taken care of */

return $\overline{c}, \overline{d}, M_{|\varphi|\times|\pi|}$

F. Example

We will use an example to describe Algorithm 4. Consider:

$$\varphi_1 = \Box x.\Diamond(((x \geq 4) \wedge s_2 \leq 5) \vee y.\Diamond((y \leq 2) \wedge s_1 \geq 0))$$

In the first step (Table III), the monitoring algorithm 4 sets the monitoring table entries of the corresponding signal predicates $\varphi_{12} : s_1 \geq 0$ and $\varphi_8 : s_2 \leq 5$, then, for the first freeze variable y, ComputeInitialTimeConstraint will evaluate the time constraint $y \leq 2$ corresponding to the instantiation of y to $\tau_0 = 0$. After that, the main algorithm calls ComputeRobustness to compute the robustness values of the frozen subformulas of the subtree $\mathrm{SubTree}_{\varphi_1}(y) = \{\varphi_9, \varphi_{10}, \varphi_{11}, \varphi_{12}\}$.

φ_i \ τ_i	0	1	2	3	4	5
$\varphi_8 = s_2 \leq 5$	-3	2	4	-1	1	-6
$\varphi_9 = \Diamond\varphi_{10}$	2	-1	-1	F	F	F
$\varphi_{10} = \varphi_{11} \wedge \varphi_{12}$	2	-2	-1	F	F	F
$\varphi_{11} = y \leq 2$	T	T	T	F	F	F
$\varphi_{12} = s_1 \geq 0$	2	-2	-1	3	-4	7

TABLE III: Monitoring table after instantiating y to $\tau_0 = 0$ (only $\mathrm{SubTree}_{\varphi_1}(y)$ entries shown).

Then, in Table IV, y will be instantiated to $\tau_1 = 1$ and the time constraint row is updated by calculating $y - 1 \leq 2$. Note here, our algorithm will not recalculate the value of the time constraint for each time sample between τ_1 and τ_5 (as in [12]). It will only calculate one value corresponding to the time sample τ_3 since it knows that the remaining values will not change. Then, the values of the subformulas of the subtree $\mathrm{SubTree}_{\varphi}(y)$ are updated for each time sample between τ_1 and τ_5 by calling ComputeSubFormula. Again, our monitoring algorithm will only recalculate the values that can change. Note that values marked in blue in Table IV are the values calculated for this instantiation.

φ_i \ τ_i	0	1	2	3	4	5
$\varphi_8 = s_2 \leq 5$	-3	2	4	-1	1	-6
$\varphi_9 = \Diamond\varphi_{10}$	2	3	3	3	F	F
$\varphi_{10} = \varphi_{11} \wedge \varphi_{12}$	2	-2	-1	3	F	F
$\varphi_{11} = y \leq 2$	T	T	T	T	F	F
$\varphi_{12} = s_1 \geq 0$	2	-2	-1	3	-4	7

TABLE IV: Monitoring table after instantiating y to $\tau_1 = 1$.

Similarly, the TSTL_1 monitor algorithm will follow the same steps for the instantiations of y to τ_2 and τ_3, and for the remaining instantiations (τ_4 and τ_5), there will be no changes in the time constraint (when the algorithm finishes with the instantiation τ_3, the row corresponding to the time constraint φ_{11} will only have T's and will remain like that for τ_4 and τ_5) and the algorithm will not be calculating any values (exit condition for the $while$ loop in line 9 of the TSTL_1 algorithm), the monitoring table will remain the same as in the instantiation of y to τ_3.

Once the robustness value of the root frozen subformulas is resolved for each timestamp, this row is copied to the parent to be used by higher level subformulas (Lines 13 and 14). In our example, φ_9 will be copied into φ_6 (Table V).

φ_i \ τ_i	0	1	2	3	4	5
$\varphi_6 = y.\varphi_9$	2	3	3	7	7	7
$\varphi_7 = x \geq 4$						
$\varphi_8 = s_2 \leq 5$	-3	2	4	-1	1	-6
$\varphi_9 = \Diamond\varphi_{10}$	2	3	3	7	7	7
$\varphi_{10} = \varphi_{11} \wedge \varphi_{12}$	2	-2	-1	3	-4	7
$\varphi_{11} = y \leq 2$	T	T	T	T	T	T
$\varphi_{12} = s_1 \geq 0$	2	-2	-1	3	-4	7

TABLE V: Monitoring table after the final instantiation of y to $\tau_5 = 5$, and copying entries for the root of $\mathrm{SubTree}_{\varphi}(y)$ to its parent, to start processing on $\mathrm{SubTree}_{\varphi_1}(x)$.

For the second and final iteration of the For loop in line 3 ($k = 2 = |V|$) of the TSTL_1 monitor algorithm, the algorithm will go through the same steps but with the variable x this time in the subtree $\mathrm{SubTree}_{\varphi}(x)$.

Finally, the TSTL_1 monitor algorithm computes the robustness value of the highest set of subformulas $\mathrm{TopSubTree}(\varphi)$ using lines 18-21. Table VI shows the final monitoring table snapshot after finishing with all instantiations of x, copying entries for the root of $\mathrm{SubTree}_{\varphi}(x)$ to its parent and evaluating the $\mathrm{TopSubTree}(\varphi) = \varphi_1$. The TSTL_1 monitor algorithm returns the first value of the first row (indicated in brown).

φ_i \ τ_i	0	1	2	3	4	5
$\varphi_1 = \Box\varphi_2$	7	7	7	7	7	7
$\varphi_2 = x.\varphi_3$	8	7	7	7	7	7
$\varphi_3 = \Diamond\varphi_4$	8	7	7	7	7	7
$\varphi_4 = \varphi_5 \vee \varphi_6$	2	3	3	7	7	7
$\varphi_5 = \varphi_7 \wedge \varphi_8$	F	F	F	F	F	F
$\varphi_6 = y.\varphi_9$	2	3	3	7	7	7
$\varphi_7 = x \geq 4$	F	F	F	F	F	F
$\varphi_8 = s_2 \leq 5$	-3	2	4	-1	8	-6
$\varphi_9 = \Diamond\varphi_{10}$	2	3	3	7	7	7
$\varphi_{10} = \varphi_{11} \wedge \varphi_{12}$	2	-2	-1	3	-4	7
$\varphi_{11} = y \leq 2$	T	T	T	T	T	T
$\varphi_{12} = s_1 \geq 0$	2	-2	-1	3	-4	7

TABLE VI: Final snapshot of the monitoring table

G. Technical Details

This section presents technical details for *ComputeInitialTimeConstraint*, *ComputeTimeConstraint*, and *ComputeSubFormula* that were skipped in Subsections V-C, V-D, and V-E.

1) Technical details: ComputeInitialTimeConstraint (Algorithm 6): This algorithm corresponds to the first instantiation of freeze variable x_k to 0. It evaluates whole time constraints rows (for each τ_i for $i \in [0, |\pi| - 1]$) into TRUE/FALSE. For each time constraint, Algorithm 6 evaluates the corresponding row depending on the time comparison operator. It takes as inputs the formula φ, the time variable x_k, the trace π and the monitoring table $M_{|\varphi|\times|\pi|}$ and outputs the vector $\overline{\alpha}$ where α_j is the position of the timestamp where the time constraint φ_j changed its value from TRUE to FALSE or the opposite (for example, if the value of a time constraint φ_9 is FALSE for the first 5 timestamps and TRUE for the rest than $\alpha_9 \leftarrow 5$).

Algorithm 6: ComputeInitialTimeConstraint

Input: $\varphi, x_k, \pi, M_{|\varphi|\times|\pi|}$
Output: $\overline{\alpha}, M_{|\varphi|\times|\pi|}$
$\overline{\alpha} \leftarrow 0$
for *each* $\varphi_j \equiv x_k \sim r$ *where j is the index of* $x_k \sim r$ *in* M **do**
  switch $\sim$ **do**
    case $<$ $(resp. \leq)$ **do**
      while $\tau_{\alpha_j} - \tau_0 < (resp. \leq)\ r$ **do**
        $M[j, \alpha_j] \leftarrow T$; $\alpha_j \leftarrow \alpha_j + 1$
      for $i \leftarrow \alpha_j$ *to* $|\pi| - 1$ **do** $M[j, i] \leftarrow F$
    case $>$ $(resp. \geq)$ **do**
      while $\tau_{\alpha_j} - \tau_0 \leq (resp. <)\ r$ **do**
        $M[j, \alpha_j] \leftarrow F$; $\alpha_j \leftarrow \alpha_j + 1$
      for $i \leftarrow \alpha_j$ *to* $|\pi| - 1$ **do** $M[j, i] \leftarrow T$
    case $=$ **do**
      while $\tau_{\alpha_j} - \tau_0 < r$ **do**
        $M[j, \alpha_j] \leftarrow F$; $\alpha_j \leftarrow \alpha_j + 1$
      if $\tau_{\alpha_j} - \tau_0 = r$ **then** $M[j, \alpha_j] \leftarrow T$
      for $i \leftarrow \alpha_j + 1$ *to* $|\pi| - 1$ **do** $M[j, i] \leftarrow F$
return $\overline{\alpha}, M_{|\varphi|\times|\pi|}$

2) Technical details: ComputeTimeConstraint (Algorithm 7): The inputs are: formula φ, a freeze variable x_k, trace π, monitoring table $M_{|\varphi|\times|\pi|}$, and timestamp position integer t ($t \geq 1$) where τ_t can be seen as the environment value $\mathcal{E}(x_k := \tau_t)$ for the instantiation τ_t of freeze variable x_k, and a vector $\overline{\alpha}$. This algorithm evaluates the values of time constraints rows for each τ_i for $i \in [t, |\pi| - 1]$). In fact, it only recalculates the values that changed from the previous instantiation of x_k for a given time constraint and returns the new value of $\overline{\alpha}$.

- case $\varphi_j = x_k \leq r$ (respectively $\varphi_j = x_k < r$): we are sure that the condition $x_k - \tau_t \leq r$ (resp. $x_k - \tau_t < r$) will remain TRUE if it was TRUE in the previous instantiation of x_k. In fact, $x_k - \tau_t < x_k - \tau_{t-1} \leq r$ (resp. $x_k - \tau_t < x_k - \tau_{t-1} < r$). So, we can start checking the values of the time constraint $x_k - \tau_t \leq r$ from the first timestamp (τ_{α_j}) when it was FALSE in the previous instantiation. Once $\tau_i - \tau_t > r$ (resp. $\tau_i - \tau_t \geq r$), we can stop because we are sure that the remaining values are already FALSE from the previous instantiation: $x_k - \tau_{t-1} > x_k - \tau_t > r$ (resp. $x_k - \tau_{t-1} > x_k - \tau_t \geq r$).
- case $\varphi_j = x_k = r$: we are sure that the condition $x_k - \tau_t = r$ will remain FALSE if $x_k - \tau_{t-1} < r$ in the previous instantiation of x. In fact, $x_k - \tau_t < x_k - \tau_{t-1} < r$. So, we can start checking the values of the time constraint $x - \tau_t = r$ from the last timestamp (τ_{α_j}) where $x_k - \tau_{t-1}$ is no longer less than r.
  Once $x_k - \tau_t > r$, we can stop because we are sure that the remaining values are already FALSE from the previous instantiation ($x_k - \tau_{t-1} > x_k - \tau_t > r$).
- case $\varphi_j = x_k \geq r$ or $x_k > r$: similar to the first case.

Algorithm 7: ComputeTimeConstraint

Input: $\varphi, x_k, \pi, t, M_{|\varphi|\times|\pi|}, \overline{\alpha}$
Output: $\overline{\alpha}, M_{|\varphi|\times|\pi|}$
for *each* $\varphi_j \equiv x_k \sim r$ *where j is the index of* $x_k \sim r$ *in* M **do**
  switch $\sim$ **do**
    case $<$ $(resp. \leq)$ **do**
      while $\tau_{\alpha_j} - \tau_t < (resp. \leq)\ r$ **do**
        $M[j, \alpha_j] \leftarrow T$; $\alpha_j \leftarrow \alpha_j + 1$
    case $>$ $(resp. \geq)$ **do**
      while $\tau_{\alpha_j} - \tau_t \leq (resp. <)\ r$ **do**
        $M[j, \alpha_j] \leftarrow F$; $\alpha_j \leftarrow \alpha_j + 1$
    case $=$ **do**
      while $\tau_{\alpha_j} - \tau_t < r$ **do**
        $M[j, \alpha_j] \leftarrow F$; $\alpha_j \leftarrow \alpha_j + 1$
      if $\tau_{\alpha_j} - \tau_t = r$ **then** $M[j, \alpha_j] \leftarrow T$
return $\overline{\alpha}, M_{|\varphi|\times|\pi|}$

3) Technical details: ComputeSubFormula (Algorithm 5): Inputs are a subformula φ_j, t ($t \geq 1$) that represents the timestamp index of the current time binding of x_k, a monitoring table $M_{|\varphi|\times|\pi|}$ and 2 vectors $\overline{c}$ and $\overline{d}$. The variable d_j is the position of the largest timestamp where the subformula φ_j changed its values from TRUE to FALSE (or the opposite) in the current instantiation of the freeze variable compared to the previous one and c_j is the position of the smallest timestamp where the subformula φ_j changed its values from TRUE to FALSE (or the opposite) in the current instantiation of the freeze variable compared to the previous one. ComputeSubformula ensures $M[j, i]$ has the correct robustness value $\rho(\varphi_j, \pi, i, \mathcal{E})$ for i in range $[t, |\pi| - 1]$. In order to do this, it leverages the work done in ComputeSubformula for φ_j in the previous instantiation of x, i.e., for the environment $\mathcal{E}[x = \tau_{t-1}]$. In the worst case, it may need to *update* the $M[j,i]$ table entries (that contained values based on the previous instantiations $(\pi, i, \mathcal{E}[x = \tau_{t-1}] \models \varphi_j)$ from index τ_t till τ_{d_j} (we are guaranteed that the robustness values of φ_j for the timestamps τ_i for $i \in [\tau_{d_j}, \tau_{|\pi|-1}]$ will not change). However, in practice fewer entries in the table need to be updated since we use $while$ loops in (lines 14 and 21) ComputeSubFormula as an early exit condition.

- If φ_j is $\varphi_m \wedge \varphi_n$ or $\varphi_m \vee \varphi_n$ (resp. $\neg\varphi_m$): these Boolean operators depend only on the current timestamp, so, in order to calculate the robustness value of φ_j at a timestamp τ_i, the algorithm only needs the values of the subformulas φ_m and φ_n (resp. φ_m) at τ_i. The algorithm uses the interval $[\min(c_m, c_n), \max(d_m, d_n)]$ (resp. $[c_m, d_m]$) to keep track of where the robustness values of φ_m and φ_n (resp. φ_m) could have changed in the current instantiation of x_k. The robustness value of φ_j can change only inside that interval. The algorithm updates the values of φ_j between τ_{c_j} and τ_{d_j} and the new interval τ_{c_j} and τ_{d_j} can be used later when ComputeSubFormula is called again for another $\varphi_{j'}$ in the current instantiation of x_k.
- If φ_j is $\varphi_m \mathcal{U} \varphi_n$ (resp. $\Box\varphi_m$ or $\Diamond\varphi_m$): For these operators,
  fact: The value of the formula φ_j at τ_i depends on the values of φ_m and φ_n (resp. φ_m) at the current timestamp τ_i and the value of φ_j at the following timestamp τ_{i+1}. Same as the previous operators, the algorithm will calculate the values of φ_j for the timestamps $[\tau_{c_j}\ ,\ \tau_{d_j}]$ and then proceeds to check the value of φ_j at τ_{c_j-1}: if the robustness value of φ_j changes compared to value of the previous instantiation of x_k to τ_{t-1}, $\rho(\varphi_j, \pi, c_j - 1, \mathcal{E}[x := \tau_t]) \neq \rho(\varphi_j, \pi, c_j - 1, \mathcal{E}[x := \tau_{t-1}])$, the algorithm will update it and decrement $c_j \leftarrow c_j - 1$, and proceeds to check the value of φ_j at the new τ_{c_j-1} and so on until the value of φ_j does not change at some point. If the value did not change, we know by the **fact**, that the values of the subformula φ_j for all the remaining timestamps τ_i for i in $[t, c_j]$ will not change as well. Otherwise, if it changes every time, then the algorithm will stop at τ_t and that is the worst case scenario.

Once the algorithm is done updating the values of a subformula φ_j, it returns the vectors $\overline{c}$ and $\overline{d}$ to be used for the remaining subformulas in $\mathrm{SubTree}_\varphi(x_k)$.

VI. Running Time of Algorithms

A. TSTL *Monitor Algorithm*

The complexity of TSTL Monitor algorithm is

$$O(|\,\mathrm{SubTree}\,\varphi(x_{|V|})| \cdot |\pi|^{|V|+1}))$$

where $|\,\mathrm{SubTree}\,\varphi(x_{|V|}|)$ is the size of the $\mathrm{SubTree}\,\varphi(x_{|V|})$, $|V|$ is the number of freeze variables and $|\pi|$ is the number of timestamps in the trace π. In fact, each instantiation of the freeze variable $x_{|V|}$ takes $O(|\,\mathrm{SubTree}\,\varphi(x_{|V|})|\cdot|\pi|))$ (lines 7-9 in Rec-TSTL) and we have $|\pi|-1-t = O(|\pi|)$ instantiations (line 1 in Rec-TSTL). And, Rec-TSTL $(|V|, t)$ (for any t) is called $|\pi|^{|V|-1}$ times.

B. TSTL_1 *Monitor Algorithm*

Let the input timed word be π, and the TSTL_1 formula φ. Suppose $r_{\min}, r_{\max}$ are the minimum and maximum time constants in φ. The upper bound of the time complexity of the monitoring algorithm is

$$O\big(|\varphi| \cdot \beta_{\max} \cdot (|\pi| - \beta_{\min})\big), \tag{4}$$

where $|\varphi|$ is the total number of subformulas in the TSTL formula ($|\varphi|$ can also be seen as the number of rows of the monitoring table), $\beta_{\max}$ denotes the maximum number of timestamps in any time interval of duration $r_{\max} + 1$ within π, and $\beta_{\min}$ denotes the minimum number of timestamps in any $r_{\min} - 1$ duration time interval within π. For the special case where φ does not have any time constraints, the complexity can be reduced to $O(|\varphi| \cdot |\pi|)$.

The complexity can also be expressed in terms of the $\mathrm{SubTree}_\varphi(x_k)$ subtrees as

$$O\left(|V| \cdot \max_k \big((|\,\mathrm{SubTree}_\varphi(x_k)| \cdot \beta^k_{\max}) \cdot (|\pi| - \beta^k_{\min})\big)\right)$$

where $|V|$ is the number of time variables, $|\,\mathrm{SubTree}_\varphi(x_k)|$ is the number of subformulas in subtree $\mathrm{SubTree}_\varphi(x_k)$ and $\beta^k_{\max}, \beta^k_{\min}$ are the same as before, but this time exclusive for the time variable x_k. Our use of the variables $\overline{c}$, $\overline{d}$ and the $while$ condition in line 9 of Algorithm 4 (the TSTL_1 Monitor Algorithm) ensures the lowering of the complexity from $|\pi^2|$ to $\beta^k\max \cdot \big(|\pi| - \beta^k_{\min}\big)$. In addition, our algorithm uses the following heuristic to reduce the running time further – we use a $while$ loop for early exit in the loops of Algorithm ComputeSubFormula. In the worst case, ComputeSubFormula for each instantiation $x_k = \tau_j$ may have to update $\beta^k_{\max}$ entries from columns j through $j + \beta^k_{\max}$. However, using the checks in the $while$ loop at lines 14 and 21 in ComputeSubFormula, the algorithm goes through a possibly reduced number of columns (depending on π and φ) for each instantiation $x_k = \tau_j$, instead of all the columns of the monitoring table.

Note that ComputeRobustness is $O(1)$ complexity, ComputeInitialTimeConstraint is $O(|\pi|)$ complexity for a single time constraint (each case of the switch statement has $|\pi|$ iterations), the complexity of ComputeTimeConstraint with the $while$ loop in line 9 of the main algorithm for a single time constraint is $O(|\pi| - \beta^k_{\min}))$ (ComputeTimeConstraint will update $(|\pi| - \beta^k_{\min})$ values of the time constraint row at most.

Finally, the complexity of ComputeSubFormula is $O(\beta^k_{\max})$: each case of the switch statement can have at most $\beta^k_{\max} = d_j - t + 1$ iterations (for the x_k instantiation $x_k = \tau_t$). In fact, the algorithm loops from d_j down to t in the worst case (that is the maximum number of iterations which can be seen in the for loop and the $while$ loop combined in Lines 10-16 or 17-23). Originally, the loop is from d_j down to c_j, however the value of c_j can decrease but it will never go lower than t, also, the value of d_j does not change or increase. The number of iterations in ComputeSubFormula will always be less or equal to $\beta^k_{\max}$. In practice, ComputeSubFormula will go through a much more lower numbers of iterations.

For the main TSTL monitor algorithm:

- The complexity of lines 4-7 is $O(|\,\mathrm{SubTree}_\varphi(x_k)| \cdot |\pi|)$ as follows: line 4 is $O(|\pi|)$, we have $|\pi|$ iterations in line 5 and $O(|\,\mathrm{SubTree}_\varphi(x_k)|)$ for lines 6 and 7 combined.
- The complexity of lines 9-15 is $O\left(|\,\mathrm{SubTree}_\varphi(x_k)| \cdot \beta^k_{\max}\right) \cdot \left(|\pi| - \beta^k_{\min}\right)\big)$ as follows: we have $(|\pi| - \beta^k_{\min})$ iterations in line 9 and $O(|\,\mathrm{SubTree}_\varphi(x_k)| \cdot \beta^k_{\max})$ for lines 13 and 14.

- The complexity of lines 16-17 is $O(|\pi|)$

Considering the for loop in line 3, and since lines 9-15 has the highest complexity inside this loop, it brings us to a $O\left(|V| \cdot \max_k \left(|\operatorname{SubTree}_{\varphi}(x_k)| \cdot \beta^k_{\max} \cdot \left(|\pi| - \beta^k_{\min}\right)\right)\right)$ complexity for lines 3-17 combined and that is the full algorithm complexity (the complexity of lines 18-21 is only $O(|\operatorname{TopSubTree}(\varphi)| \cdot |\pi|)$).

VII. Experiments

Our experiments were conducted on a 64-bit Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz with 16-GB RAM and we implemented all algorithms in Matlab, without using any special libraries and compared the performance. The traces used are uniformly sampled with a sampling rate of 1 second, the signal predicates values are randomly generated to integer values in $[-50, 50]$, and each experiment is executed 10 times with 10 different generated traces in order to obtain the mean and variance values. As the timestamps are integers $0, 1, \ldots |\pi| - 1$, the constants $\beta_{\min}, \beta_{\max}$ in the algorithm running time bounds can be taken to be $r_{\min} - 1, r_{\max} + 1$, where $r_{\min}, r_{\max}$ are the minimum and maximum constants in the formulas.

In Table VII, we run the TSTL monitor algorithm on 4 different TSTL formulas with different number of freeze variables and different length of traces. The formulas are:

- $\varphi_1 = x.(s_1 \geq 2 \to \Diamond(s_2 > 3 \wedge \Diamond(s_3 > 1 \wedge x \leq 5)))$
- $\varphi_2 = x.(s_1 \geq 2 \to \Diamond(s_2 > 3 \wedge \Box(x \leq 5 \to s_3 > 1)))$
- $\varphi_3 = x.(s_1 \geq 2 \to \Diamond(s_2 > 3 \wedge y.\Diamond(s_3 > 1 \wedge x \leq 5 \wedge y \leq 2)))$ [Running example]
- $\varphi_4 = x.(s_1 \geq 2 \to \Diamond(s_2 > 3 \wedge y.\Box((x \leq 5 \wedge y \leq 2) \to s_3 > 1)))$

		$\lvert\pi\rvert = 100$		$\lvert\pi\rvert = 200$		$\lvert\pi\rvert = 500$	
φ	$\lvert V\rvert$	Mean	Var	Mean	Var	Mean	Var
φ_1	1	3.18	0.01	12.42	0.01	86.31	0.52
φ_2	1	3.16	0.01	12.33	0.02	85.83	0.59
φ_3	2	65.92	0.27	654.3	1.70	–	–
φ_4	2	67.17	0.37	661.2	1.91	–	–

TABLE VII: Running times for different $|\pi|$ and $|\varphi|$ values.

As we discussed in the complexity section, the experiment results prove that the time complexity of the TSTL monitor algorithm that we presented is exponential to the number of freeze variables. Once we exceed two freeze variables and we use a large trace, this algorithm will no longer be useful. For $|\pi| = 500$, we encountered a timeout for φ_3 and φ_4 after one hour and 5 minutes.

In Table VIII, we run the TSTL$_1$ monitor algorithm on 7 different TSTL$_1$ formulas with different number of freeze variables and different number of subformulas. For a formula ϕ, we define the number of subformulas $|\phi|$ as the number of operators in ϕ ($\Box, \Diamond$ and $\to$ each count as one operator) plus the number of time constraints ($x \in [a, b]$ count as three: $x \geq a \wedge x \leq b$). The number of subformula does not include the number of signal predicates $s \sim r$. For example $|\phi_1|$ below is equal to 3: $x > 8$, $\to$ and $\Box$. The running time variances for $|\pi| = 1000$ and $|\pi| = 2500$ are ≤ 0.01. The formulas are:

- $\phi_1 = \Box x.(x > 8 \to s_1 \leq 10)$
- $\phi_2 = x.(s_1 > -5 \to \Box(s_2 \geq 0 \wedge \Diamond(s_3 > 5 \,\mathcal{U}\, x \leq 12)))$
- $\phi_3 = x.(s_1 > 7 \to \Box(s_2 \geq 0 \wedge x \geq 4) \wedge (\Diamond y.(s_3 > 0 \,\mathcal{U}\, y \leq 8)))$
- $\phi_4 = \Box x.(y.(s_1 > 2 \to \Diamond(s_2 > 5 \wedge y \leq 4)) \wedge \Diamond(s_3 < 0 \wedge x \leq 12))$
- $\phi_5 = z.((x.(s_1 < 20 \to \Box(\neg s_2 \leq 20 \wedge x \leq 4)) \wedge \Diamond y.(s_3 \geq 10 \,\mathcal{U}\, y \leq 8)) \wedge z \leq 12)$
- $\phi_6 = z.(x.(s_1 \leq 15 \to \Box(s_2 \leq 20 \wedge x \leq 4) \wedge \Diamond y.(s_3 \geq 10 \,\mathcal{U}\, y \leq 8)) \vee \Box w.(s_4 \geq 5 \wedge w \geq 5)) \wedge z \leq 12)$
- $\phi_7 = \Diamond z.(x.(\neg s_1 > 3 \to \Box(s_2 \geq 2 \wedge x \leq 4) \wedge \Diamond y.(s_1 > 16 \wedge s_3 \geq -50 \,\mathcal{U}\, y \leq 8)) \vee \Box w.(s_4 \geq 5 \wedge w \geq 5)) \wedge (s_5 \geq 0 \wedge z \leq 12))$

		$\lvert\pi\rvert = 1000$	$\lvert\pi\rvert = 2500$	$\lvert\pi\rvert = 5000$		$\lvert\pi\rvert = 10000$	
ϕ	$\lvert\phi\rvert$	Mean	Mean	Mean	Var	Mean	Var
ϕ_1	3	0.84	2.01	4.06	0.01	8.23	0.04
ϕ_2	6	2.96	7.43	14.97	0.03	29.40	0.02
ϕ_3	8	3.51	8.91	17.63	0.06	34.91	0.02
ϕ_4	9	3.48	8.55	17.12	0.04	34.31	0.02
ϕ_5	11	4.31	10.60	21.31	0.06	42.77	0.04
ϕ_6	14	5.68	14.32	28.71	0.03	57.91	0.05
ϕ_7	18	7.25	18.09	36.31	0.02	72.76	0.11

TABLE VIII: Running times for different $|\pi|$ and $|\phi|$ values.

The obtained experimental results conforms with our complexity analysis and our algorithm proves to be practical for industrial applications with its running time which is linear in the length of the trace. We also see that the TSTL$_1$ algorithm far outperforms the general TSTL algorithm over TSTL$_1$ formulae.

In Table IX, we use the same TSTL$_1$ formula with different time constraint intervals. The formulas are :

- $\theta_1 = \Diamond y.(y > 50 \wedge \Box(x.(x \in [10, 20] \to s_1 \geq 0)))$
- $\theta_2 = \Diamond y.(y > 50 \wedge \Box(x.(x \in [20, 30] \to s_1 \geq 0)))$
- $\theta_3 = \Diamond y.(y > 50 \wedge \Box(x.(x \in [10, 30] \to s_1 \geq 0)))$
- $\theta_4 = \Diamond y.(y > 50 \wedge \Box(x.(x \in [10, 50] \to s_1 \geq 0)))$

	$\lvert\pi\rvert = 1000$	$\lvert\pi\rvert = 2500$	$\lvert\pi\rvert = 5000$		$\lvert\pi\rvert = 10000$	
θ	Mean	Mean	Mean	Var	Mean	Var
θ_1	3.08	7.80	15.77	0.01	31.30	0.04
θ_2	3.10	7.79	15.84	0.01	31.51	0.05
θ_3	5.47	13.42	26.97	0.02	54.86	0.04
θ_4	8.69	22.20	44.45	0.05	89.07	0.09

TABLE IX: Different time constraint interval results.

Table IX shows that, since the intervals in the time constraint directly affects the length of the interval $[\tau_{\min(\overline{c})}, \tau_{\max(\overline{d})}]$ in our algorithm, the bigger the interval in the time constraint the higher the running time. In fact, an interval is basically two time constraints φ_1 and φ_2 with $\wedge$ operator: $\varphi_j = \varphi_1 \wedge \varphi_2$ and when updating the time constraints values using ComputeTimeConstraint, this algorithm returns positions α_j where each time constraint flips its value. Since, c_j and d_j are initiated respectively to $\min(c_1, c_2)$ and $\max(d_1, d_2)$, their values have a dependence on the time constraint interval length. We know that our algorithm has to calculate all the values between $[\tau_{c_j}, \tau_{d_j}]$ for every instantiation and that explains the running times shown in Table IX. We can see, based on the results from θ_1 and θ_2, that for the same interval length, we basically have the same running time, even though the largest formula constant in θ_2 is larger.

VIII. Conclusion

In this work we developed a quantitative theory of robustness for TPTL over signals. This logic, TSTL, with the presence of time-freeze quantifiers can express many natural engineering properties that STL cannot. We explored properties of our developed robustness function, such as how it can be used to identify "bad" formulae which are identically true or false purely due to the time constraints in the formula, irrespective of the signal values (Proposition 1), and the connection to the Boolean semantic (Theorem 1). We also developed two robustness computation algorithms. The first, for general TSTL formulae, and the second for a fragment –

TSTL_1– which is also more expressive than STL. This second algorithm is *linear* in the size of the trace (assuming a bounded sampling skew), and thus can be employed over long traces. Our algorithms avoid the use of complicated data structures or libraries, and instead rely on careful use of divide and conquer, and dynamic programming techniques to work over a table to compute the robustness values. We envision the use of simple data structures will facilitate efficient development and deployment on diverse platforms. Our present work and developed algorithms open up use of TSTL specifications for industrial Cyber-Physical Systems in black-box optimization based falsification approaches.

References

[1] T. Akazaki and I. Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *CAV'15*, LNCS 9207, pages 356–374. Springer, 2015.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[3] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[4] T. Anevlavis, M. Philippe, D. Neider, and P. Tabuada. Being correct is not enough: Efficient verification using robust linear temporal logic. *ACM Trans. Comput. Log.*, 23(2):8:1–8:39, 2022.

[5] C. Baier and J. P. Katoen. *Principles of model checking*. MIT Press, 2008.

[6] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010.

[7] P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In *EMSOFT'02*, LNCS, pages 294–304. Springer, 2002.

[8] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

[9] K. Chatterjee and V. S. Prabhu. Quantitative temporal simulation and refinement distances for timed systems. *IEEE Trans. Autom. Control.*, 60(9):2291–2306, 2015.

[10] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia. Robust online monitoring of signal temporal logic. *Formal Methods Syst. Des.*, 51(1):5–30, 2017.

[11] V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

[12] A. Dokhanchi, B. Hoxha, C. E. Tuncali, and G. Fainekos. An efficient algorithm for monitoring practical TPTL specifications. In *MEMOCODE'16*, pages 184–193. IEEE, 2016.

[13] A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *CAV'13*, LNCS 8044, pages 264–279. Springer, 2013.

[14] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, LNCS, pages 92–106. Springer, 2010.

[15] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA'15*, LNCS, pages 127–142. Springer, 2015.

[16] G. Ernst, S. Sedwards, Z. Zhang, and I. Hasuo. Falsification of hybrid systems using adaptive probabilistic search. *ACM Trans. Model. Comput. Simul.*, 31(3):18:1–18:22, 2021.

[17] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[18] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using s-taliro. In *ACC'12*, pages 3567–3572. IEEE, 2012.

[19] B. Ghorbel and V.S. Prabhu. Linear time monitoring for one variable TPTL. In *HSCC '22*. ACM, 2022.

[20] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Tran. on Aut. Control*, 52(5):782–798, 2007.

[21] A. Grez, F. Mazowiecki, M. Pilipczuk, G. Puppis, and C. Riveros. Dynamic data structures for timed automata acceptance. In *IPEC' 21*, volume 214 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[22] T. A. Henzinger. Quantitative reactive modeling and verification. *Comput. Sci. Res. Dev.*, 28(4):331–344, 2013.

[23] T. A. Henzinger and N. E. Saraç. Quantitative and approximate monitoring. In *LICS'21*, pages 1–14. IEEE, 2021.

[24] Y. Hirshfeld and A. Rabinovich. Expressiveness of metric modalities for continuous time. *Log. Methods Comput. Sci.*, 3(1), 2007.

[25] P. Hunter, J. Ouaknine, and J. Worrell. Expressive completeness for metric temporal logic. In *LICS'13*, pages 349–357. IEEE, 2013.

[26] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. R. Butts. Powertrain control verification benchmark. In *HSCC'14*, pages 253–262. ACM, 2014.

[27] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, 2016.

[28] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[29] Z-K. Liu and Y. Yao. *Robust Control: Theory and Applications*. Wiley Publishing, 1st edition, 2016.

[30] R. Majumdar and V. S. Prabhu. Computing distances between reach flowpipes. In *HSCC' 16*, pages 267–276. ACM, 2016.

[31] O. Maler and D. Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.

[32] N. Markey and J-F. Raskin. Model checking restricted sets of timed paths. *Theor. Comput. Sci.*, 358(2-3):273–292, 2006.

[33] D. Nickovic and N. Piterman. From mtl to deterministic timed automata. In *FORMATS' 10*, LNCS 6246, pages 152–167. Springer, 2010.

[34] P. K. Pandya and S. S. Shah. On expressive powers of timed logics: Comparing boundedness, non-punctuality, and deterministic freezing. In *CONCUR'11*, volume 6901 of *LNCS*, pages 60–75. Springer, 2011.

[35] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. Model predictive control for signal temporal logic specification. *CoRR*, abs/1703.09563, 2017.

[36] G. Rosu and K. Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, 2001.

[37] S. Sankaranarayanan and G. E. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *HSCC'12*, pages 125–134. ACM, 2012.

[38] P. Tabuada, S. Y. Caliskan, M. Rungger, and R. Majumdar. Towards robustness for cyber-physical systems. *IEEE Trans. Autom. Control.*, 59(12):3151–3163, 2014.

[39] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

[40] M. Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In *HSCC'20*, pages 11:1–11:13. ACM, 2020.

[41] M. Waga and É. André. Online parametric timed pattern matching with automata-based skipping. In *NFM' 19*, LNCS 11460, pages 371–389. Springer, 2019.

[42] M. Waga, I. Hasuo, and K. Suenaga. Efficient online timed pattern matching by automata-based skipping. In *FORMATS' 17, Proceedings*, LNCS 10419, pages 224–243. Springer, 2017.

[43] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini, and I. Hasuo. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Trans. on CAD of Int. Circuits & Systems*, 37(11):2894–2905, 2018.

Bassem Ghorbel received the polyvalent engineering degree in 2021 from Tunisia Polytechnic School, Tunis, Tunisia. Currently, he is a Computer Science Ph.D. student at Colorado State University. His research interests include formal methods, model checking, and temporal logics.

Vinayak Prabhu obtained the Ph.D. degree in the department of Electrical Engineering and Computer Sciences at the University of California, Berkeley (USA), and is currently with the Computer Science department at Colorado State University. Dr. Prabhu's research interests are in modelling, analysis, verification, security, and control of reactive, real-time, hybrid, and multi-agent systems.